MLP-verkon arkkitehtuurin optimointi geneettisellä algoritmilla

Paavo Toivanen

Tampereen yliopisto
Informaatiotieteiden yksikkö/tko
Pro gradu -tutkielma
Ohjaaja: Martti Juhola
13.5.2013

Tampereen yliopisto
Informaatiotieteiden yksikkö/tko
Paavo Toivanen: MLP-verkon arkkitehtuurin optimointi
geneettisellä algoritmilla
Pro gradu -tutkielma, 62 sivua
Toukokuu 2013

Tiivistelmä

Tämä tutkielma käsittelee eteenpäin syöttävän monikerroksisen neuroverkon (en. Multilayer perceptron, MLP) rakenteellista optimointia luokittelutehtävää varten.

Tutkielmassa esitellään Java-ympäristössä algoritmi, joka tekee haun oppimisaineiston hyvin luokittelevan verkkorakenteen löytämiseksi. Optimoidun tuloksen oikeellisuutta tutkitaan tilastollisin menetelmin. Algoritmi on hajautettu, eli se mahdollistaa verkon rakenteen optimoinnin klusteroiduilla tai internetin yli jaetuilla resursseilla.

Algoritmia sovelletaan lääketieteellisiin aineistoihin sekä generoituihin datajoukkoihin.

Avainsanat ja -sanonnat: neuroverkot, ohjattu oppiminen, geneettiset algoritmit.

CR-luokat: F.1.1, G.3, I.2.6, I.2.8

Sisällys

1 Johdanto

Keinotekoiset neuroverkot (engl. artificial neural networks, ANN) tai lyhemmin *neuroverkot* on joukko laskennallisia työkaluja, joita käytetään erilaisissa päätös- ja ennustustehtävissä. Sovelluskohteita tekniikan ja tieteen alalla ovat esimerkiksi hahmontunnistus ja signaalinkäsittely, erilaiset kontrolliongelmat sekä kognitio- ja lääketieteen, kemian, psykologian ja kielitieteen teorianmuodostus ja mallinnustehtävät.

Tässä tutkielmassa tarkastellaan erään suositun neuroverkkoarkkitehtuurin, *MLP-verkon* (engl. multilayer perceptron), käyttämistä *luokittelutehtävissä*. MLP-verkon määrittäminen tiettyä tehtävää varten on haaste, joka koskee yhtäältä verkolle annettavan tehtävän eli opetusdatan määrittelyä, toisaalta oikeiden rakenteellisten parametrien ja oppimisparametrien löytämistä verkolle sekä verkon koneellista opettamista.

Tutkielman tutkimusongelmana on MLP-verkon rakenteen optimointi. Päämääränä on löytää yhtäältä joustava algoritmi, joka etsii sopivan rakenteen tunnettuun luokitteluongelmaan, ja toisaalta esittää perustelu, miksi algoritmin löytämää rakennetta voi pitää hyvänä. Opetettavaksi tarkoitettu otanta opetusdataa ja viitteelliset oikeat tulokset tunnetaan. Optimointi määritellään hakutehtäväksi diskreetissä arkkitehtuuriavaruudessa siten, että haku etsii hyvän verkkorakenne–oppimisalgoritmiyhdistelmän.

Tämä johdantoluku jäsentelee neuroverkkoihin liittyvän käsitteistön, jota tutkielmassa käytetään, ja pohjustaa toisessa ja kolmannessa luvussa tapahtuvaa tarkempaa ongelmanmäärittelyä.

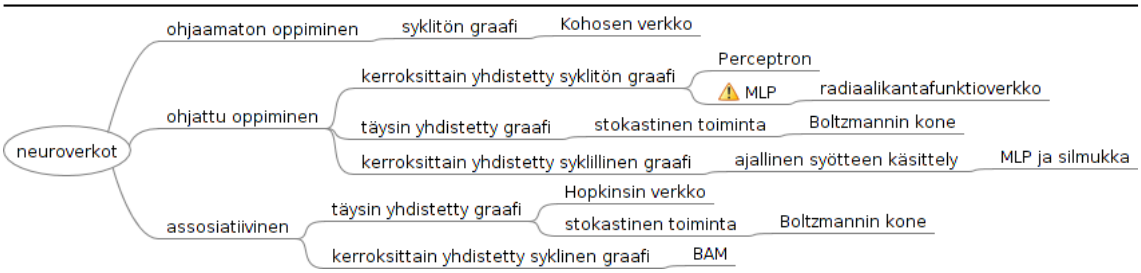
Esimerkeissä hyödynnän Java-kielellä kehittämääni neurolaskentakirjastoa, jonka lähdekoodiin viittaan online-linkein.[1] Java-esimerkkien ja sanallisen kuvauksen lisäksi käytän paikoin myös hieman formaalimpaa matemaattista esitystä, osin yhdenmukaisuuden saavuttamiseksi tutkimuskirjallisuuden kanssa. Joissakin kohdin olen yksinkertaistanut tarpeettoman kaavamaisina pitämiäni esitystapoja.

---

[1] Ks. `https://github.com/pvto/neur`. Viittaan Java-lähdekoodiin nimetyin linkein, esimerkiksi näin: [J-EBP]. Referoitu lähdekoodi on listattu luvussa 6 taulukossa 6.1. Lukijaa pyydetään huomioimaan, että lähdekoodin tyyli poikkeaa hieman Oraclen perinteisestä Java Code Convention -suosituksesta (`http://www.oracle.com/technetwork/java/codeconventions-150003.pdf`). Erityisesti luokkamuuttujat on kätketty vain siinä tapauksessa, että niitä ei haluta lukea asiakaskoodista käsin. Luokissa suositaan mahdollisuuksien mukaan funktionaalisesta ohjelmoinnista tuttua tilatonta lähestymistapaa, mutta missä tämä ei ole mahdollista tai järkevää, on joko olion metodien tai julkisten muuttujien määrä minimoitu. Oliot on suunniteltu Java-virtuaalikoneessa yhdestä säikeestä käsin käytettäviksi, ellei dokumentaatiossa muuta mainita.

Neuroverkot hyödyntävät idealtaan *paralleelilaskentaa* (engl. parallel processing), eli käsittelevät tiedon *hajautettua esitystä* (distributed representation) yhtäaikaisesti useissa erillisissä toisiinsa kytköksissä olevissa yksiköissä. [Rumelhart & McClelland, 1986]. Neuroverkoille ominaista ja vierasta monille supertietokonearkkitehtuureille, jotka voivat käyttää jopa kymmeniätuhansia yhtäaikaisia prosessoreja, on laskentayksiköiden äärimmäinen yksinkertaisuus [Tosic, 2004]. Neuroverkkoja on rakennettu sekä fyysisistä komponenteista että ohjelmistoina [Wasserman, 1989, Hopfield, 1982].

Kuvassa 1.1 on jäsennelty MLP-verkon luonnetta. Kuvan hierarkia ei ole täy-



Kuva 1.1 Neuroverkkoja. Lyhenteistä BAM tarkoittaa kaksisuuntaista assosiatiivimuistia, MLP monikerroksista perceptron-verkkoa, jota käsitellään tässä tutkielmassa.

sin absoluuttinen, vaan esimerkiksi siinä esitetyistä verkoista Boltzmannin konetta voidaan soveltaa sekä ohjatussa oppimisessa että ongelmanratkaisussa ilman opettamista.

Keskeistä monille neuroverkkomalleille on oppimisen idea, eli näkemys neuroverkosta tilallisena järjestelmänä, jota voidaan muokata *oppimisalgoritmien* (learning algorithm) avulla [Minsky & Papert, 1969]. Tällöin puhutaan *koneoppimisesta* (machine learning).

Ohjatussa oppimisessa (supervised learning) verkkoa opetetaan syöte–vastearvoparien perusteella [Mitchel, 1997]. Verkko esimerkiksi opetetaan liittämään kuva A-kirjaimesta (syöte) symboliin A (vaste) ja kuva B-kirjaimesta (syöte) symboliin B (vaste). Toinen koneoppimisparadigma on *ohjaamaton oppiminen* (unsupervised learning), jossa verkko itse muodostaa ontologian eli jäsennyksen maailmasta syötedatan perusteella [emt.]. Eräs ensimmäisistä ohjaamattoman oppimisen sovelluksista oli Teuvo Kohosen [1982] kehittämä itseorganisoituvan neuroverkon malli, jossa avaruudellisesti järjestettyyn hilaan muodostuu datan tilastollisia piirteitä vastaavia

keskuksia. Tämä tutkielma ei käsittele ohjaamatonta oppimista.

Neuroverkot voidaan jakaa hyväksymänsä syötteen ja toimintansa pohjalta *kausaalisiin* ja *ei-kausaalisiin verkkoihin*. Kausaalisilla verkoilla käsitellyissä *aikasarjoissa* ajallista riippuvuutta mallinnetaan verkon rakenteessa ja informaation syklimäisessä liikkumisessa verkon sisällä. Tällöin verkon saamien perättäisten syötteiden välille ymmärretään kausaalinen riippuvuus, jota verkon sisältämä syklisyys mallintaa. Esimerkkejä ovat esimerkiksi puhesignaalin foneettinen jäsentäminen tai pörssikurssin ennustaminen. Tämä tutkielma ei käsittele kausaalisia neuroverkkoja.

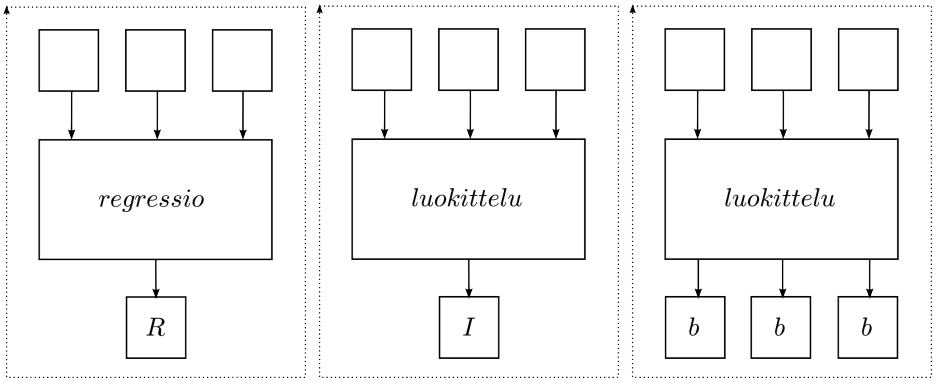
Ei-kausaaliset verkot käsittelevät perättäisiä syötteitä, joiden välillä ei ole riippuvuutta. Syötedatan oletetaan noudattavan jotakin tuntematonta mallia, jonka neuroverkko voi omaksua perehdyttyään opetusaineistoon riittävän perusteellisesti. Konkreettisesti ajateltuna opetusaineiston käsittävä *datajoukko* (data set) koostuu sarakkeellisista riveistä. A- ja B-kirjaimia luokitteleva neuroverkko voi käsitellä kuvan pikselin arvoa muuttujana sen loogisen position – pikselin koordinaatin – suhteen, jolloin kuva sisältää yhtä monta syötesaraketta kuin siinä on pikseleitä. Verkon suorituskyvyn kannalta voi toki olla tärkeää rajoittaa syötteenä annettujen pikselien määrää. Kaikki hahmontunnistustehtävät eivät liioin onnistu ei-kausaalisilla verkoilla, vaan muodontunnistus voi olla helpompaa kausaalisin menetelmin, esimerkiksi syöttämällä verkolle kuva riveittäin ja sarakkeittain [Duda et al., 2001]. Ei-kausaalinen MLP-verkko on *eteenpäin syöttävä*, mitä käsitellään alempana.

Kausaalinen–ei-kausaalinen -jaottelun lisäksi voidaan erotella joukko rajatapauksia, joissa ei-kausaalisella verkolla tehdään jossakin määrin kausaalista analyysiä. Tämä onnistuu rakenteistamalla verkolle syötetty data *aikaikkunaksi*. Esimerkiksi opetusjoukko koostuu riveistä, joiden sarakkeissa on mittaustulokset auringon aktiivisuudesta kolmenkymmenen perättäisen vuoden aikana, ja verkko saa tehtäväkseen ennustaa 31. vuoden tuloksen. [2]

Ei-kausaalista dataa käsittelevät neuroverkot suorittavat joko *regressio-* tai *luokittelutehtäviä* [Puma-Villanueva et al., 2012, Duda et al., 2001]. Luokittelutehtävässä verkko yhdistää syötteen johonkin ennalta määrättyyn luokkaan. Tällainen neuroverkko toteuttaa vaikkapa kuvauksen $f : \mathbb{R}^i \to \{1, 2, ..., o\}$, jossa i on syötevektorin koko ja o kohdeluokkien lukumäärä, tai kuvauksen $f : \mathbb{R}^i \to \{0, 1\}^o$, jossa o on tulosvektorin koko. Jälkimmäisellä rakenteella luokittelun opettaminen verkolle on helpompaa kuin ensimmäisellä, jos luokkia on useita [Duda et al., 2001]. Regres-

[2] Ks. `http://neuroph.svn.sourceforge.net/viewvc/neuroph/trunk/neuroph-2.7/Contrib/src/main/java/org/neuroph/contrib/samples/SunSpots.java?revision=1508&view=markup`

siotehtävässä neuroverkon tuloste on reaalinen ja kuvaus tyypillisesti $f : \mathbb{R}^i \to \mathbb{R}$. Tällöin neuroverkko toimii yleisluontoisesti approksimointivälineenä [Duda et al., 2001], ja vielä siinä erityisessä mielessä, että approksimoitava malli ei välttämättä ole kiinnitetty tai sitä ei täysin tunneta (vrt. [Rissanen, 1996]). Erilaisia verkkoja



Kuva 1.2 Erilaisia neuroverkkoja tulosteen perusteella jäsennettynä. Regressiota suorittava verkko tulostaa reaalisen arvon (R) ja luokitteleva verkko puolestaan kokonaisluvun (I), jonka arvo merkitsee tiettyä luokkaa. Edellistä tyypillisempi luokitteleva verkko (oikealla) tulostaa useita binäärisiä arvoja, joista kukin merkitsee yhtä luokkaa.

havainnollistetaan kuvassa 1.2.

Neuroverkon soveltuminen tiettyyn ongelmaan, parhaan neuroverkkotyypin ja sen rakenteen määrittäminen ovat tapauskohtaisia kysymyksiä. Tilanteesta riippuen neuroverkkoa luontevampi voi olla esimerkiksi tilastollinen malli tai sääntöpohjainen asiantuntijajärjestelmä. Aikakriittisissä tehtävissä teknisiin laskelmiin perustuva deterministinen ohjelma voi suoriutua neuroverkkoa paremmin sekä tehokkuuden että luotettavuuden kannalta. Kilpailevia koneoppimismenetelmiä ovat esimerkiksi *tukivektorikoneet* (support vector machine, SVM), *Bayesilaiset luokittelijat* (bayesian classifiers) ja *Markovin piilomallit* (hidden Markov models). Kyky käsitellä kompleksisia syötteitä auttaa neuroverkkoa interpoloimaan eli antaa sille rajallista kykyä käsitellä oikein syötteitä, joita sille ei ole koskaan opetettu, sekä virhesietoisuutta kykynä käsitellä syöte oikein huolimatta sen sisältämästä lievästä kohinasta. Matemaattiseen informaatioteoriaan liittyviä kysymyksiä ei käsitellä syvällisesti tässä opinnäytetyössä.

1900-luvun puolivälissä kehitetyt yksikerroksiset perceptron-verkot kykenivät lineaarisiin erotteluihin, eivät esimerkiksi loogisen xor-kuvauksen mallintamiseen [Minsky & Papert, 1969]. Yksikerroksiseen perceptron-verkkoon pohjautuva monikerrok-

sinen perceptron-verkko rakentuu signaalia jalostavista *kerroksista*. Mikäli kerroksia on vähintään kolme eli verkko sisältää syöte- ja tuloskerrosten lisäksi vähintään yhden *piilokerroksen* (hidden layer), MLP-verkko kykenee tiettyjen lisäehtojen täyttyessä epälineaarisiin erotteluihin. Tällöin sopivankokoista MLP-verkkoa voidaan opettaa n-asteisen polynomin rajaamien alueiden erottelemiseen toisistaan. Yleisesti kaikkien MLP-mallien joukko voidaan ymmärtää parametroiduksi regressiofunktioiden perheeksi. [Duda et al., 2001, Anthony, 2001, Rynkiewicz, 2012]. Oppimisalgoritmeja monikerroksisille perceptron-verkoille julkaistiin 1970- ja 1980-luvuilla (ks. [Wasserman, 1989]). Erityisesti Rumelhartin ja muiden julkaisut ovat tulleet tunnetuiksi [Rumelhart et al., 1986, Rumelhart & McClelland, 1986].

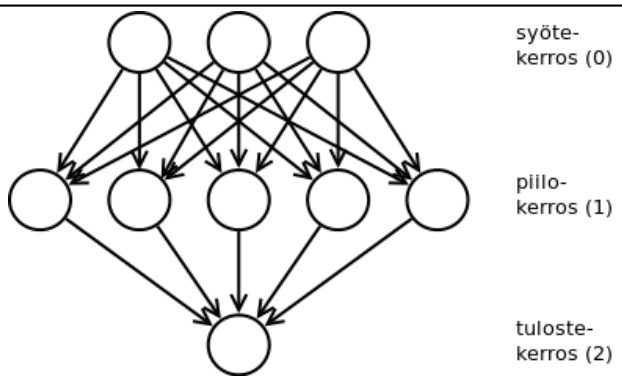
Neuroverkon määrittävä graafi on *täysin yhdistetty*, kun jokaisesta neuronista johtaa kaari jokaiseen muuhun verkon neuroniin. Esimerkiksi Boltzmannin koneen määrittelee tällainen täysin yhdistetty graafi.

MLP-verkko on yksinkertaisimmillaan *eteenpäin syöttävä* (feedforward), mikä tarkoittaa että verkon rakenteen määrittelevä graafi on syklitön eli se ei sisällä silmukoita. Syöttö- ja oppimisalgoritmit käsittelevät verkkoa kerroksina, joissa on kussakin joukko laskennallisia yksikköjä eli *neuroneja*. Tietyn kerroksen neuronit on eteenpäin syöttävässä verkossa yhdistetty vain lähinnä seuraavan kerroksen neuroneihin, ei koskaan esimerkiksi toisiinsa saman kerroksen sisällä. Kerroksittain eteenpäin syöttävässä MLP-verkossa jokaisella neuronilla on yhteys kaikkiin sitä seuraavan kerroksen neuroneihin. Syklittömyys ja täydellinen yhdistettyys kerroksittain ovat olleet merkittäviä piirteitä MLP-verkon laskennallisten piirteiden käsittelyssä, mutta ne eivät ole nykyteorian valossa välttämättömiä. Verkon ilmaisuvoiman kannalta kerrosarkkitehtuurin rikkominen voi tuottaa etuja (vrt. [Puma-Villanueva et al., 2012]).

Eteenpäin syöttävän MLP-verkon arkkitehtuuri on pohjana tälle tutkielmalle. Muita tärkeitä neuroverkkoarkkitehtuureja ovat mainittu Kohosen verkko [Kohonen, 1982], Hopfieldin verkko [Hopfield, 1982] ja Boltzmannin kone [Ackley & Hinton, 1985]. Myös tiettyjä assosiatiivimuisteihin lukeutuvia verkkoja kuten kaksisuuntaista assosiatiivimuistia (bi-directional associative memory, BAM) voidaan pitää toimintansa puolesta neuroverkkona [Juhola, 2012]. Uudemmassa kirjallisuudessa esiintyvät esimerkiksi nestetilakone (liquid state machine), kaksisuuntainen assosiatiivimuisti impulsseilla [Shao, 2012] ja evolutiivisiin algoritmeihin pohjautuva evolinoverkko [Schmidhuber et al., 2007], joka oppii esimerkiksi neljän päällekkäin asetetun sinikäyrän muodostaman haastavan regressiotehtävän.

Kuva 1.3 havainnollistaa vielä MLP-verkon rakennetta suunnattuna graafina. Kuvan verkossa on kolme kerrosta: kolmineuroninen syötekerros, viisineuroninen pii-

lokerros ja yksineuroninen tuloskerros. Kuvan jokainen kerros on täysin yhdistetty lähinnä seuraavaan. Eteenpäin syöttävässä MLP-verkossa tieto kulkee syötekerrok-



Kuva 1.3 MLP-verkon rakenne. Eteenpäin syöttävässä verkossa aktivaatio siirtyy kerroksittain aina eteenpäin. Kukin ylemmän kerroksen neuroni on MLP-verkossa yhdistetty kaikkiin lähinnä alemman kerroksen neuroneihin. Kuvan verkossa on yksi piilokerros, mutta piilokerroksia voi olla myös useita.

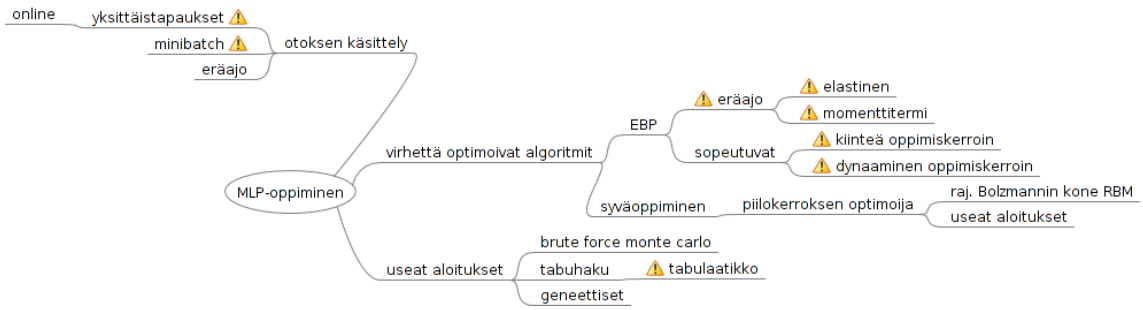
selta kerroksittain piilokerrosten läpi kohti tuloskerrosta.

Erityisen tärkeä neuroverkoille ominainen piirre on tapa, jolla ne yhdistävät tiedon ja oppimisen. Neuroverkkoa voidaan opettaa kumulatiivisesti lisäten sille uusia tehtäviä samalla kun se säilyttää kompetenssia vanhoissa jo opituissa tehtävissä. Neuroverkon opettaminen vaikuttaa sen käyttäytymiseen, eikä verkon hyödyntämiseksi välttämättä tarvitse tuntea verkolle syötetyn datan takana olevia ”todellisia” korrelaatioita (tällaisen mallioppimisen lähtökohdista ja sudenkuopista ks. [Rissanen, 1996]). MLP-verkossa tietoa varastoidaan neuronien välisiin *kaarien painoarvoihin* (connection weight).

Neuroverkkoa opetetaan *oppimislagoritmin* avulla. MLP-verkossa oppimisalgoritmit voidaan toteuttaa yksisuuntaisia kaaria hyväksi käyttäen. Rumelhartin ja muiden [1986] esittelemä *vastavirta-algoritmi* (Error Backpropagation, EBP) perustuu kaarten optimointiin verkon tuloksen ja viitearvon välisen virheen perusteella (ks. toteutus: [J-EBP]). EBP-algoritmi on virhegradienttiin perustuva algoritmi, jossa verkon yksittäisen neuronin aktivaation virhettä pienennetään iteratiivisesti. EBP ja muut virheen ”takaisin päin” siirtämiseen perustuvat algoritmit vaativat verkon kaaria opetettavan kerroksittain piilokerrokselta lähtien takaisin kohti syötekerrosta. Tämän mahdollistamiseksi joko verkon tai oppimisalgoritmin on tallennettava neuronien saamat kokonaissyötteet opettamista varten. Toteutuksessa [J-NEU] neuroni tallentaa EBP:n vaatimat parametrit. EBP-algoritmia voidaan optimoida mo-

nin erilaisin muunnoksin, joista Sarkar [1995] tekee katsauksen. Tässä tutkielmassa hyödynnetään niin sanottua *elastista* algoritmia (elastic EBP; resilient EBP), jossa kullakin verkon neuronilla on oma oppimistahtinsa [J-ELA], ja *momenttitermillistä* versiota [J-MOM], jossa edellisten oppimisepokkien oppimissuunta virhepinnalla vaikuttaa taannehtivasti oppimiseen.

MLP-verkon oppimisalgoritmeja on listattu kuvassa 1.4. Tutkielmassa käytetyt algoritmit on merkitty korostusmerkillä. MLP-verkkojen oppimisessa erotellaan kak-



Kuva 1.4 MLP-verkon oppiminen: oikealla on kuvattu oppimisalgoritmien keskinäisiä suhteita ja *otoksen käsittely*-haarassa eri menetelmiä, jotka määräävät, millä rytmillä verkko oppii.

si toisistaan poikkeavaa käytäntöä, *eräoppiminen* (batch learning) ja yksittäistapauksista oppiminen (non-batch learning). Stokastisluonteinen eräoppiminen perustuu määrättyyn otokseen dataa. Yhden oppimissyklin eli *epokin* aikana koko otos ajetaan verkon läpi ja verkkoa opetetaan tulosten keskivirheellä epokin lopuksi [J-TS].

Oppiminen yksittäistapauksista jaetaan edelleen *sopeutuvaan* (adaptive) ja *reaaliaikaiseen oppimiseen* (online learning). Verkkoa opetetaan yksittäisen syötteen syöttämisen jälkeen. Sopeutuvassa oppimisessa otos on kiinnitetty kuten eräajo-oppimisessa, ja epokin aikana verkkoa opetetaan yhtä monta kertaa kuin otoksessa on tietueita. Reaaliaikaisessa oppimisessa puolestaan otos ei ole kiinnitetty ennalta käsin, vaan verkkoa voidaan opettaa ennalta tuntemattomilla näytteillä, esimerkiksi anturin antamilla mittaustuloksilla. [Duda et al., 2001, Chen & Soo, 1995].

EBP-algoritmi takaa verkon painoarvojen siirtyvän kohti virhepinnan minimiä, jos oppimisessa käytettävät verkon muutokset ovat äärettömän pieniä [Rumelhart & McClelland, 1986]. Tämä johtaisi epäkäytännöllisen pitkään oppimisaikaan [Wasserman, 1989]. Oppimisnopeuteen ja virhealttiuteen vaikuttaa algoritmille paramet-

roitu *oppimiskerroin* (learning rate coefficient). Erä- ja sopeutuvan oppimisen käytännöt vaikuttavat siihen, suuntautuuko oppiminen kohti otoksen keskivirheen suppenemista vai heittelehtiikö yksittäisten syötteiden opettamisen myötä edestakaisin. Molemmat käytännöt johtavat joihinkin ongelmiin ja rajoituksiin otoksen suhteen [Duda et al., 2001].

Eräoppimisen ja sopeutuvan oppimisen välimuotona on pienoiserien (minibatch) käyttö. Esimerkiksi Glorot ja Bengoa [2010] käyttävät suuren kuva-aineiston opettamisessa kymmenen näytteen minibatch-eriä siten, että verkkoa opetetaan aina kymmenen kuvan keskivirheellä, muuten kuten eräajo-oppimisessa.

Syvyyssuuntautuneiden oppimiskäytäntöjen lisäksi on olemassa joukko stokastisia oppimisalgoritmeja, jotka eivät opeta verkkoa optimoimalla virhettä [Wasserman, 1989]. Helpoin tällainen algoritmi on brute force monte carlo -haku kaikkien mahdollisten verkon painoarvovektoreiden joukkoon [emt.]. *Useiden satunnaisten aloitusten menetelmissä* (multistart method) hyödynnetään verkon opettamisessa yritystä ja erehdystä ja jotakin syvyyshaun tekevää opetusalgoritmia. Verkon parametreja kehitetään useista lähtötilanteista käsin, kunnes löydetään kyllin hyväksi arvioitu verkko jatko-opetusta varten. [Oh & Pedrycz, 2005]. Muita vaihtoehtoja ovat esimerkiksi geneettisellä algoritmilla toteutetut haut [Chandra et al., 2012] ja tabuhaun tekniikoita hyödyntävät algoritmit [Oh & Pedrycz, 2005]. Tutkielmassa verkon esiopetusvaiheessa hyödynnetään tabu-hakuun [Sexton & al., 1995] ja ratkaisuavaruuden laatikoimiseen perustuvaa uutta algoritmia. Algoritmia kuvataan tarkemmin luvun 2 lopussa. *Esiopetus* tarkoittaa neuroverkon parametrointia ennen varsinaista ohjattua oppimista [Erhan et al., 2009].

Aktiivisen tutkimusalueen neuroverkkojen opettamisessa muodostavat *syvät oppimismenetelmät* (deep learning), joissa opetetaan useamman piilokerroksen sisältäviä verkkoja [Glorot & Bengio, 2010, Erhan et al., 2009]. Tutkielmassa ei käsitellä syvällisesti näitä menetelmiä, joskin monikerroksisen verkon opettamiseen poimitaan joitakin tuloksia.

Neuroverkkojärjestelmien ymmärtämiselle ja hallinnalle lisäävät dynaamisuus ja laskennan hajautuminen haasteita. Eräs kirjallisuudessa tunnettu esimerkki neuroverkon rajoituksista on tilanne, jossa verkolle syötettävän datan jakauma muuttuu ajan myötä. Perinteiset neuroverkkomallit eivät sisällä menetelmiä tällaisen ongelman havaitsemiseksi tai siihen reagoimiseksi, joten syötedatan jakauman muuttuessa neuroverkko alkaa kaikessa hiljaisuudessa antaa yhä virheellisempiä tuloksia. Ominaisuus on ongelmallinen erityisesti sen takia, että muutos voi tapahtua vähitellen, eikä verkon syötedatalle asettamia ennakkoehtoja ole välttämättä alun perin kirjattu ylös, mikäli niitä on tunnettukaan. [Juhola, 2012].

Yleensä voidaan todeta, että MLP-verkko ei ole älykäs saamansa syötteen suhteen, vaan se tarvitsee syötteensä harkitussa muodossa. Neuroverkkojärjestelmä vaatii myös edellä mainitun ja muiden ongelmien takia ylläpitoa ja valvontaa. [Juhola, 2012]. Hyvin parametroitu neuroverkko on virhesietoinen syötteen sisältämän kohinan suhteen, eli se antaa oikeansuuntaisia tuloksia myös syötteelle, jota se ei täsmällisesti tunne ennalta käsin tai joka sisältää pienen satunnaisen virheen. Inhimillisestä älykkyydestä ei vielä ole kyse, sillä virhesietoisuus ei ole sama asia kuin kyky ratkaista odottamattomia ongelmia, vaikkakin virhesietoisuutta voidaan pitää yhtenä älykkyyden osatekijänä [Russell & Norvig, 2009].

Neuroverkon rakenne vaikuttaa sen ominaisuuksiin: esimerkiksi neuronien määrä ja järjestyminen vaikuttavat verkon ilmaisuvoimaan eli siihen, miten vaikeisiin tehtäviin verkko soveltuu. Toisaalta verkon tai sen osien tapa käsitellä syötettä vaikuttaa verkon laskennallisiin ominaisuuksiin. Näitä asioita käsitellään yksityiskohtaisemmin toisessa luvussa. Seuraavaksi pohdin lyhyesti MLP-verkon toiminnallisen rakenteen formalisointia.

Lee [1991] jakaa neuroverkot dynamiikaltaan kolmelle tasolle.

1. **Toiminnallinen taso** tai *signaalin prosessoinnin taso*, jolla ainoastaan verkon neuronien tila voi muuttua. Tällä tasolla toimii Leen mukaan Hopfieldin verkko.

2. **Parametrien taso** tai *parametrinen oppimisen taso* sisältää neuronien ja kaarten painoarvojen parametroinnin. Tälle tasolle kuuluvat Leen mukaan esimerkiksi Kohosen itsejärjestyvät verkot [Kohonen, 1982] ja Boltzmannin kone [Ackley & Hinton, 1985].

3. **Rakenteellinen taso** tai *rakenteellisen kehityksen taso*, jolla on samat ominaisuudet kuin toiminnallisen ja parametrisen tason verkoilla sekä lisäksi mahdollisuus lisätä, muuttaa ja poistaa muita verkon rakenteeseen vaikuttavia parametreja kuten neuroneja ja niiden välisiä yhteyksiä. [Lee, 1991].

Leen [1991] tutkimuksesta tulee kyseenalaistaa, voiko eteenpäin syöttäviä MLP-verkkoja ja Hopfieldin verkon ja Boltzmannin koneen kaltaisia syklisiä verkkoja tarkastella samassa formaalissa viitekehyksessä, joka ei sisällä kuvausta eri verkkotyyppien toimintaperiaatteista. Rekursiivisen verkon matemaattinen perusta on erilainen kuin eteenpäin syöttävän verkon [Anthony, 2001]. Sovellan Leen sinänsä opettavaista jaottelua tässä tutkielmassa vain eteenpäin syöttävien neuroverkkojen ymmärtämiseen.

Erottelen käsitteet *verkon opettaminen* ja *verkon arkkitehtuurin optimointi*. MLP-verkon opettamiseen kuuluu verkon kaarten painoarvojen muuttaminen. Opettaminen ei määritelmällisesti siis sisällä verkon rakenteen muuttamista. *Arkkitehtuurin optimoinnilla* tarkoitan *rakenteellisen tason* piirteiden määrittämistä – ennen verkon opettamista. Optimoinnissa valitaan rakenteelliset ominaisuudet verkolle, joka ei myöhemmin menetä niitä opetuksen aikana. Kirjallisuudessa esiintyy myös dynaaminen lähestymistapa, jossa neuroverkon rakennetta optimoidaan myös oppimisen aikana [Lee, 1991]. Lähestymistavassa on teoreettisia ongelmia verkon oppimistulosten kannalta. Niiden ratkaisemiseen vaaditaan apuheuristiikan käyttöä [Puma-Villanueva et al., 2012]. Tässä tutkielmassa tarkastellaan sopivan rakenteen löytämistä vertaamalla kahta rakennetta keskenään, ei muuntamalla yksittäisen neuroverkon rakennetta.

MLP-verkon rakenne määrää sen laskennalliset ominaisuudet [Duda et al., 2001, Anthony, 2001]. Tätä käsitellään tutkielman toisessa luvussa. Tutkielman tutkimusongelma liittyy verkon arkkitehtuurin optimointiin, eli määrätyn *oppimistehtävän* ratkaisemiseen soveltuvan neuroverkon löytämiseen.

Parametrointitehtävän formalisoimiseksi voidaan määritellä tietue

$$(S, A), \tag{1.1}$$

missä S on otos opetusdataa ja A joukko mahdollisia neuroverkon rakenteita.

Parametrointifunktio

$$p(S, A) \longrightarrow A \tag{1.2}$$

antaa tulokseksi otollisen neuroverkon rakenteen $a \in A$,

$$a = (\mathbf{n}, f, L), \tag{1.3}$$

jossa vektori $\mathbf{n}$ sisältää syöte-, piilo- ja tuloskerrosten koot (> 0) ja f on verkon *aktivaatiofunktio* sekä L verkon opettamisessa käytetty oppimisalgoritmi. Parametrointifunktion kannalta kuvatut parametrit $\mathbf{n}$, f ja L koostavat joukon A määrittämän *hakuavaruuden* dimensioita.

Käytännössä tarvitaan myös yksittäisen neuroverkon sarjallistamista massamuistiin. Sarjallistamiseen tarvitaan kuvaus verkon rakenteesta sekä verkon sisäisten parametrien kuvaus: $N = (a, W)$, missä N on neuroverkko, a verkon rakenne ja W vektori, joka sisältää verkon kaarten painoarvot.

Kuvatulle täysin yhdistetylle eteenpäin syöttävälle kolmikerroksiselle verkolle painoarvovektorin koko $|W|$ on vakaa: $|W| = n_0 \cdot n_1 + n_1 \cdot n_2$. Opetusalgoritmin näkökulmasta tietty verkon rakenne määrittää *ratkaisuavaruuden*, jonka piiristä algoritmi etsii parasta mahdollista painoarvovektoria. Jos esimerkiksi verkossa on 1, 2

ja 1 syöte-, piilo- ja tulosneuronia, on painoarvovektorin koko $|W| = 1 \cdot 2 + 2 \cdot 1 = 4$, ja reaalisilla kaarten painoarvoilla varustetun verkon ratkaisuavaruus on siis neliulotteinen $\mathbb{R}^4$. Ratkaisuavaruus tulee käsitteenä erotella verkon *dimensiosta*, joka ymmärretään toisinaan verkon syöteneuronien määräksi [Anthony, 2001], mutta tässä tutkielmassa syöte- ja tulosneuronien luvuksi $m + o$.

Dimensiota ja aktivaatiofunktiota käsitellään tarkemmin toisessa luvussa, jossa myös kuvataan seikkaperäisemmin, miksi optimointitehtävään on valittu juuri kuvatut rakenteelliset piirteet.

Uudemmassa tutkimuksessa neuroverkkojen dynaamista parametrointia on lähestytty useista näkökulmista (esim. [Oh & Pedrycz, 2005, Babu & Suresh, 2012, Ballabio et al., 2011, Puma-Villanueva et al., 2012, Yang & Chen, 2012]). Verkon rakenteen optimointi geneettisten algoritmien avulla on eräs suosittu tekniikka [Maniezzo, 1994, Oh & Pedrycz, 2005, Ballabio et al., 2011].

Tutkielman rakenne on seuraava. Luvussa kaksi käsitellään MLP-verkon rakennetta ja parametrointia. Luvussa kolme käsitellään verkon arkkitehtuurin optimointia sekä esitetään eräs geneettinen algoritmi optimointiprosessin suorittamiseksi. Luvussa neljä tarkastellaan esimerkkiaineistojen valossa parametroinnin onnistumista esitetyssä viitekehyksessä. Tuloksia verrataan täydelliseen hakuun verkon parametriavaruudessa. Tällainen haku takaa parhaan mahdollisen tuloksen, johon MLP-verkko voi kyseisessä ongelmassa päästä. Lopuksi tarkastellaan tutkielman aihepiiriä ja onnistumista ja tehdään joitakin huomioita MLP-verkkojen käytöstä.

2 MLP-verkon rakenteellisia piirteitä

MLP-verkon toimintaan vaikuttavia rakenteellisia piirteitä on listattu taulukossa 2.2.

Piirre	Vaikutus
kaarten painoarvot yhtä kuin 0	verkkoa ei voi opettaa
kaarten painoarvot eroavat nollasta	opetettavuus
painoarvojen jakauma ei nollan ympärillä	opetettaessa divergoi helposti
painoarvojen jakauma U[-n,n]	verkko stabiilimpi opetettaessa
vain kaksi verkon kerrosta	vain lineaarisia erotteluja
yli kaksi kerrosta	epälineaarisia erotteluja
yli kolme kerrosta (piilokerroksia useita)	oppiminen vaikeutuu
normalisoitu painoarvojen jakauma	syvä verkko helpompi opettaa
lineaarinen aktivaatiofuntio	vain lineaarisia erotteluja
epälineaarinen aktivaatiofunktio	epälineaarisia erotteluja
. . .	. . .
aktivaatiofuntiona sigmoidinen funktio	viitearvojen normalisointi [0,1]
aktivaatiofuntiona hyperbolinen tangentti tai softsign	viitearvojen normalisointi [-1,1]
. . .	. . .
aktivaatiofunktion jyrkkyys	oppimisnopeuden vaihtelu; oppimisalgoritmin toimivuus
tuloskerros luokkien määrää harvempi	vaikeampi opettaa
tuloskerros vastaa luokkien määrää	suppenee helpommin ratkaisuun
liian vähän piilokerroksen neuroneja	epätarkat ennusteet
liikaa piilokerroksen neuroneja	ylisovitus
otos liian pieni	ylisovitus, epätarkat ennusteet
otos hyvin suuri	opettaminen hidasta
oppimisalgoritmi epätarkka	divergoi helpommin pois globaalista ratkaisusta
oppimisalgoritmi hyvin tarkka	oppiminen hidasta
EBP ja eräajo-oppiminen	kokonaisoppimisvirhe suppenee; algoritmin toimivuus; luokkajakauma vaikuttaa voimakkaasti oppimistulokseen
sopeutuva EBP	opetusjärjestys merkitsee; algoritmin toimivuus

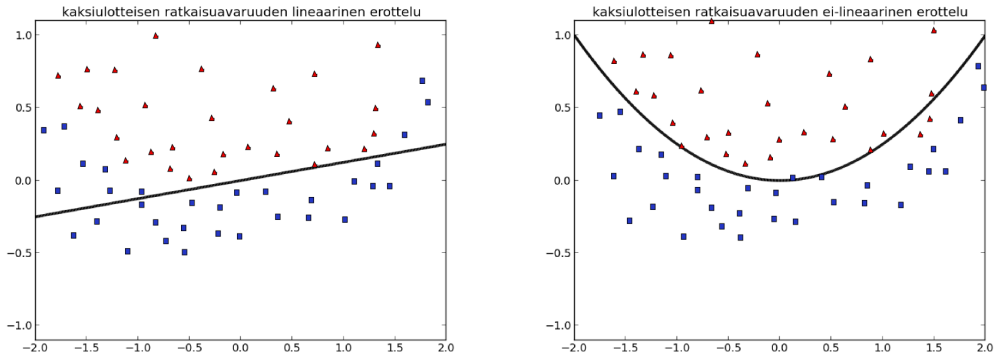
Taulukko 2.2: Luokittelevan MLP-verkon rakenteelliset piirteet

MLP-verkossa tietoa varastoidaan kaarten painoarvoihin, joita puolestaan opetetaan opetusalgoritmeillä. Virhettä iteratiivisesti optimoivat algoritmit toimivat, jos verkko toimii, eli silloin kun kaarten painoarvot eroavat nollasta [Duda et al., 2001]. Jos painoarvojen keskiarvo poikkeaa nollasta, saa *virhegradienttiin* perustuva opetusalgoritmi (kuten EBP) painoarvot helposti harhautumaan niin, että ne opetuksen myötä kasvavat hallitsemattomasti samalla kun verkko juuttuu toivottomaan tilaan opetusalgoritmin kannalta. Ongelman välttää helposti alustamalla kaarten painoarvot satunnaisluvuilla tasajakaumasta $U[-n, n]$. Vastaavaa ongelmaa ei esiinny stokastisia oppimisalgoritmeja käytettäessä. [Wasserman, 1989]. Perinteisesti ta-

sajakauma tietylle kerrokselle tulevien kaarten painoarvoille on määrätty suhteessa syöttävien neuronien määrään [Erhan et al., 2009]:

$$U\left[\frac{-1}{\sqrt{k}}, \frac{1}{\sqrt{k}}\right]. \tag{2.1}$$

MLP-verkossa tulee olla yksi tai useampi piilokerros, jotta verkon suorittamat erottelut voivat olla monimutkaisempia kuin *lineaarisia* ratkaisuavaruudessaan [Minsky & Papert, 1969]. Kuva 2.1 havainnollistaa *lineaarista* ja *epälineaarista erottelua* kaksiulotteisessa ratkaisuavaruudessa. Ratkaisuavaruuden moniulotteisuus eli dimensio vastaa neuroverkon syöte- ja tulosneuronien yhteenlaskettua määrää $|\mathbf{x}| + |\mathbf{z}|$, missä $\mathbf{x}$ on syötekerroksen neuronien ja $\mathbf{z}$ vastaavasti tuloskerroksen neuronien muodostama vektori.[1]



Kuva 2.1 Erotteluja kaksiulotteisessa ratkaisuavaruudessa.

MLP-verkkoon voidaan rakentaa niin monta piilokerrosta kuin halutaan. Tämä tutkielma rajoittuu kolmi–kuusikerroksisiin verkkoihin [Erhan et al., 2009]. Kolmogorovin teoreeman mukaan useilla piilokerroksilla varustettu verkko on teoreettisen ilmaisuvoiman kannalta samankaltainen kuin kolmikerroksinen, eli useampikerroksinen verkko ei opi erotteluja, joita jokin kolmikerroksinen verkko ei myös opi [Wasserman, 1989, Duda et al., 2001]. Syvän verkon opettamista EBP-algoritmilla on pidetty hitaampana ja algoritmin ja verkon lähtötilanteen valinnan suhteen sitä vaativampana, mitä enemmän kerroksia verkossa on [Duda et al., 2001]. Hinton ja muut [2006] käyttävät erillistä piilokerrosten optimoijaa tällaisten syvien MLP-verkkojen (deep belief nets) opettamisen helpottamiseksi. Erhanin ja muiden [2009]

[1] Tässä puhutaan verkon ratkaisuavaruudesta, kun taas ensimmäisessä luvussa puhuttiin opetusalgoritmin ratkaisuavaruudesta.

mukaan kompleksisuusanalyysi tukee syvien rakenteiden käyttöä monissa tilanteissa, sillä matalampi rakenne voi vaatia eksponentiaalisen määrän piiloneuroneita. Toisaalta syvän arkkitehtuurin opettaminen sisältää mahdollisesti ratkeamattoman ei-konveksisen optimointiongelman [emt.]. Glorot'n ja Bengion [2010] mukaan syvän verkon esiopetuksessa haetaan tasapainoa verkossa eteenpäin (syöttö) ja taaksepäin (opetus) suuntautuvan läpivirtauksen maksimoimiseksi. Glorot'n ja Bengion mukaan optimaalista on alustaa *normalisoidut painoarvot* (normalised weights) tasajakaumasta

$$U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right], \tag{2.2}$$

jossa n_j on neuronille tulevan painoarvovektorin koko ja n_{j+1} siltä lähtevän painoarvovektorin koko.

Aktivaatiofunktiolla (activation function) tarkoitetaan funktiota, joka säätelee sitä, millä tapaa neuroni välittää toisilta neuroneilta saamaansa *syötettä aktivaationa* eteenpäin. Rumelhart ja McClelland [1986] käsittelevät aktivaatiofunktiota neuronin ja koko verkon yhteisenä jaettuna piirteenä ja käyttävät epälineaarista derivoituvaa *rajafuntiota* (threshold function), esimerkiksi *hyberbolista tangenttia* $tanh(x)$ tai jotakin *sigmoidista funktioita* (sigmoid function)

$$\sigma(x) = \frac{1}{1 + e^{-kx}}, \tag{2.3}$$

jossa k on positiivinen vakio, joka säätelee sigmoidisen funktion *jyrkkyyttä* (steepness) eli sitä, miten lähellä se on muodoltaan *askelfunktiota* (step function)

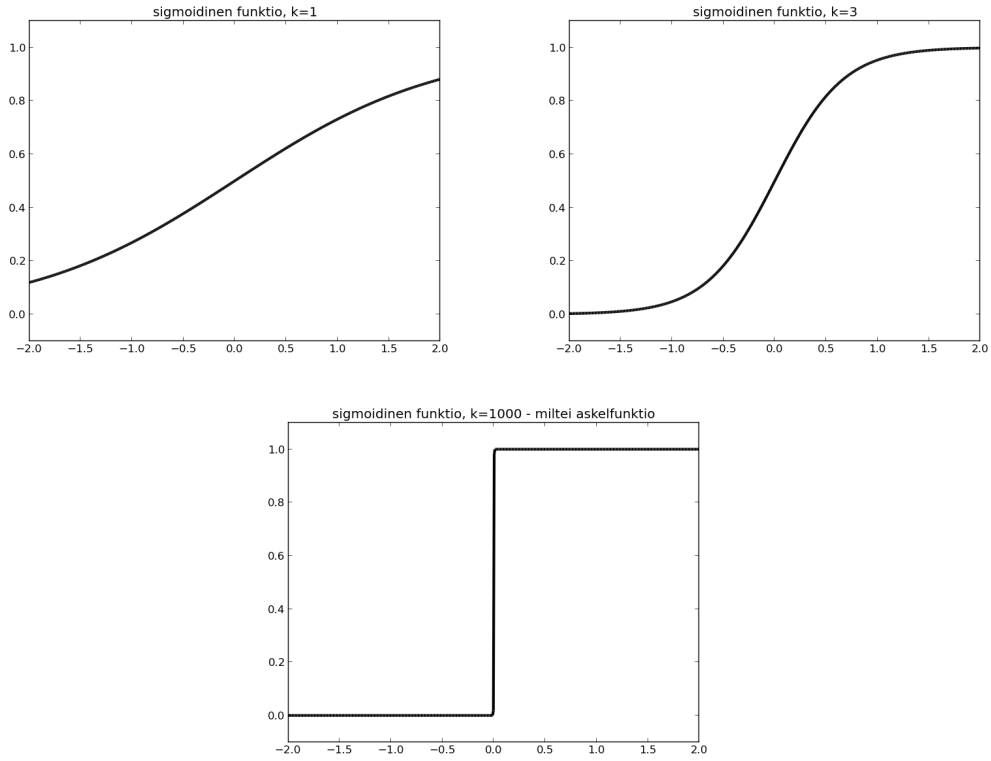
$$a(x) = \begin{cases} 1 & \text{kun } x \geq 0 \\ 0 & \text{kun } x < 0 \end{cases}. \tag{2.4}$$

Myös askelfunktio on eräs yksinkertainen rajafunktio [Wasserman, 1989]. Poikkeuksena epälineaarisuudelle on syötekerros, jolta aktivaatio syötetään yleensä lineaarisesti seuraavalle kerrokselle [emt.]. Mitä jyrkempi aktivaatiofunktio on, sitä nopeammin oppiminen tapahtuu tietyssä vaiheessa virhepintaa ja vastaavasti sitä nopeammin neuroverkon neuronit erikoistuvat opetettaessa. Uudemmassa neuroverkkokirjallisuudessa aktivaatiofunktion vaihtelu on usein vapaampaa verkon sisällä. Esimerkiksi radiaalikantafunktioverkossa ja sumeassa polynomisessa verkossa yksittäisen neuronin aktivaatiofunktiota optimoidaan verkon opettamisen aikana [Duda et al., 2001, Oh & Pedrycz, 2005]).

Softsign-funktio on Bergstran ja muiden [2009] ehdottama verrattain tuore aktivaatiofunktio, jonka tehokkuudesta koneoppimisessa Glorot ja muut [2010] anta-

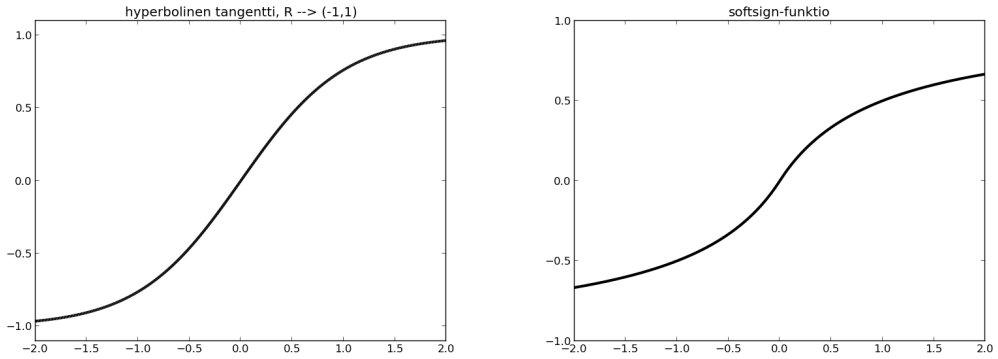
vat lupaavia näyttöjä. Verrattuna tanh-funktioon softsign-funktion hännät lähestyvät asymptoottejaan hitaammin, mikä johtaa Glorot'n ja muiden mukaan parempaan konvergenssiin oppimisessa kauempana oppimistehtävän virhepinnan minimistä. Tanh- ja softsignfunktiot välttävät sigmoidisessa funktiossa ongelmallisen epäsymmetrisyyden x-akselin suhteen [Bergstra & al., 2009].

Sigmoidinen ja askelfunktio esitetään kuvassa 2.2 ja hyperbolinen tangentti sekä softsign-funktio kuvassa 2.3.



Kuva 2.2 Sigmoidiset funktiot.

Aktivaatiofunktion derivoituvuus on ennakkoehtona *virhegradienttia* käyttäville opetusalgoritmeille [Rumelhart & McClelland, 1986] (ks. luku 3). Kun aktivaatiofuntio(t) tunnetaan, voidaan verkon muodostama kuvaus myös formalisoida. Aktivaatiofunktion on oltava epälineaarinen – muussa tapauksessa verkon antama kuvaus on palautettavissa lineaariseksi operaatioksi (kuten matriisien kertolaskussa), jolloin verkon toiminta regressoituu kaksikerroksista perceptron-verkkoa vastaavaksi [Duda et al., 2001]. Oppimisen kannalta on myös huomioitava tulostekerroksen neuronien aktivaatiofunktio: viitteelliset vastearvot pitää normalisoida funktion arvojoukon mukaisesti esimerkiksi välille [0,1] (sigmoidinen funktio) tai [-1,1] (hyperbolinen



Kuva 2.3 Tanh- ja softsign-funktiot.

tangentti).

Eräänä opetuksen erityisongelmana voidaan pitää opetusalgoritmin juuttumista pitkille hyvin pienen derivaatan alueille aktivaatiofunktiossa, jolloin oppiminen hidastuu tai pysähtyy kokonaan. Ongelman helpottamiseksi voidaan lisätä funktion derivaattaan sitä vääristävä vakiosyöte, joka auttaa oppimista katvealueilla:

$$f'^{*} = f' + \lambda. \tag{2.5}$$

Kynnys on tyypillisesti pieni. Tutkielmassa käytetään kiinteää arvoa $\lambda = 0.1$.

Aktivaatiofunktion lisäksi kirjallisuudessa eritellään myös neuronin syötefunktio, joka määrää, miten toisilta neuroneilta tuleva syöte kootaan kokonaissyötteeksi, joka sitten syötetään aktivaatiofunktiolle. Tutkielman piirissä käytetään yleistä syötefunktiota, joka summaa neuronille j tulevat syötteet:

$$\sigma_j = \sum (f_i \cdot w_{ij}). \tag{2.6}$$

Tässä neuronilta i neuronille j tulevan kaaren painoarvo w_{ij} kerrotaan syöttävän neuronin aktivaatiolla f_i ja neuroni j saa syötteekseen sille muilta neuroneilta tulevien aktivaatioiden painotetun summan.

Luokittelutehtävässä luokat voidaan kiinnittää yksittäisen neuronin antaman palautteen eri arvoiksi. Varsinkin jos luokkia on useita tällainen rakenne voi aiheuttaa ongelman verkon opettamisessa: kaaret eivät opetuksen aikana pääse erikoistumaan mihinkään ongelma-avaruuden piirteisiin eikä verkko tunnu oppivan uutta informaatiota. Opetuksen menestyksekkyyttä lisää luokkien erotteleminen erillisille tuloskerroksen neuroneille. [Juhola, 2012]. Tutkielmassa käytetään optimoitua *softmax*-luokittelijafunktiota $c : (\mathbf{z}, i) \longrightarrow \{0, 1\}$, joka antaa arvon 1 vain tuloskerroksella suurimman aktivaation saaneelle neuronille:

$$c(\mathbf{z}, i) = \begin{cases} 1 & \text{kun } \mathbf{z}[i] > \mathbf{z}[j] \\ 0 & \text{muuten} \end{cases} , j \in \{1, .., o\} \setminus i. \tag{2.7}$$

Luokittelijafunktion avulla saadaan viitearvoihin sovitettu tulosvektori **z'**,

$$\mathbf{z'} = (c(\mathbf{z}, 1), c(\mathbf{z}, 2), \ldots, c(\mathbf{z}, o)), \tag{2.8}$$

missä o on tuloskerroksen neuronien ja luokkien määrä. Sovitetun tulosvektorin ja viitearvovektorin perusteella saadaan virhettä kuvaava energiafunktio verkon kerrokselle. Tätä käsitellään vielä tässä luvussa tarkemmin.

Piilokerroksen koko voi olla sekä liian suuri että liian pieni ongelmaan nähden. Yleisesti mitä enemmän piilokerroksen neuroneita on, sen monimutkaisemman erottelun ratkaisemiseen verkko sopii. Ongelma on aina yhteydessä opetettavaan dataan ja sitä ei siis tule abstrahoida tyystin datan ulkopuolelle [Rissanen, 1996].

Näin yhden piilokerroksen sisältävän MLP-verkon ilmaisuvoima on verrannollinen sekä piilokerroksilla sijaitsevien neuronien lukumäärään että opetettavan otoksen suuruuteen [Duda et al., 2001]. Tarkemmin ottaen verkon piiloneuronien määrän ja verkolle opetettavan otoksen suuruuden väliltä on löydetty erilaisia korrelaatioita. Erityisesti jos verkon piilokerroksen neuronien määrää ei rajoiteta, johtaa tämä vahvaan ylisovittumiseen gaussilaista kohinaa sisältävällä datalla, vaikka otos olisi suurikin [Rynkiewicz, 2012].

Korrelaatio neuronien määrän ja opittavan aineiston koon välillä tunnetaan useilla verkkotyypeillä – esimerkiksi Hopfieldin verkon oppimien kuvioiden määrä ei ylitä verkon neuronien määrää [Wasserman, 1989]. Kohosen itseoppivan verkon suhteen vastaava ilmiö on intuitiivisesti ymmärrettävä, koska hilan yksittäiset neuronit edustavat opetettavan datan piirteiden keskuksia [Kohonen, 1982].

Ylisovittumiseksi (overfitting) kutsutaan tilannetta, jossa neuroverkko oppii antamaan oikeita tuloksia opetusaineistonsa (*opetusjoukko*) syötteille, mutta erehtyy helposti opetuksessa tapaamattomien syötteiden (*testijoukko*) ollessa kyseessä [Rynkiewicz, 2012]. Tässä oletetaan, että molemmat syötejoukot ovat *hyviä* eli tulevat samasta jakaumasta, vaikka voivatkin sisältää satunnaisia virheitä eli *kohinaa*. Ylisovittumiseen johtaa tilanne, jossa verkon piilokerroksella on enemmän neuroneita kuin on tarpeen ongelman ratkaisevan yleistyksen muodostamiseksi datasta. Vähäiset piilokerroksen neuronit johtavat puolestaan siihen, että verkon muodostamien erottelujen sisältämät piirteet ovat nekin vähiä.

Opetettavan otoksen suhteen voidaan havaita rinnakkaisia ongelmia edellisten kanssa. Mikäli opetettava otos on liian pieni, johtaa tämä helposti ylisovittumiseen, koska oikeanlaista yleistystä on vaikea päätellä datasta. Jos taas opetettava otos

on hyvin suuri eikä kohina ole liian suurta, on yleistyksen päätteleminen mahdollista. Monimutkaista ongelmaa varten voidaan kuitenkin tarvita paljon piilokerroksen neuroneja, ja verkon opettaminen hidastuu sekä datamäärän kasvaessa että verkon koon kasvaessa. Otoksen näytteet voivat myös opetettaessa vetää opetusalgoritmia yhdessä "väärään" suuntaan, jolloin verkon näytteiden perusteella muodostama malli ei esimerkiksi vastaa generoivaa jakaumaa. Tämä voi johtua sekä näytteiden *harvuudesta* (sparsity of data) että niiden tiheästä sijoittumisesta toisiinsa nähden.

Aina yritykset löytää opetusalgoritmin avulla ratkaisu opetettavaan ongelmaan eivät pääty onnistumiseen. Tällöin voi olla kyseessä, kuten edellä, verkon divergoiminen, tai myös opetusalgoritmin juuttuminen *paikalliseen minimiin* ratkaisuavaruudessa. Jälkimmäisestä ilmiöstä Duda [2001] tekee havainnollisen käsittelyn. Ratkaistava ongelma voi myös olla ratkeamaton tai satunnaisesti ratkeava riippuen verkon painoarvoista ja opetusalgoritmista. Ongelman ja opetusalgoritmin suhdetta on tarkasteltu PAC-mallissa (en. probably approximately correct) [Valiant, 1984], ks. myös [Anthony, 2001].

Opetusalgoritmin ja oppimistehtävän suhteella on vaikutusta verkkorakenteen suorituskykyyn. Yleisessä tilanteessa ei voida varmistua edeltä käsin siitä, toimiiko jokin oppimisalgoritmi tietylle oppimistehtävälle. Samalla verkkorakenteella tehdyissä luokitteluissa on keskinäistä satunnaista vaihtelua, joka aiheutuu yhtäältä siitä, että verkko alustetaan satunnaisluvuilla ja toisaalta myös siitä, jos opetusalgoritmissa on epädeterministinen elementti.

Neuronin virhe e kuvaa sen antaman tuloksen sisältämää virhettä viitearvoon nähden:

$$e = z - \alpha, \tag{2.9}$$

missä z on neuronin aktivaatio ja α neuronin viitearvo (tuloskerroksen neuronille opetusvektorista ja piilokerroksen neuronille EBP-algoritmissa koostettuna verkon alemmalta kerrokselta [Rumelhart et al., 1986]).

Näytteen prosessoinnin jälkeen verkon antamaa tulosta määrittää *energiafunktio* E, joka kuvaa verkon tekemän luokittelun sisältämän virheen suuruutta. Rumelhart ja muut [1986] esittelevät erään energiafunktion, *keskineliövirheen* (mean square error, MSE), jota voidaan käyttää useilla erilaisilla aktivaatiofunktioilla:

$$E(z) = \frac{1}{2}\frac{\sum e^2}{o}, \tag{2.10}$$

missä o on tulosarvovektorin koko.

Energiafunktio kuvaa myös opetusalgoritmin antaman ratkaisun hyvyyttä ope-

tuksen jossakin vaiheessa. Otantana toimii joko *opetusjoukko* T tai *testijoukko* V, jotka molemmat tulevat samasta datajoukosta eli otoksesta S. Tässä tutkielmassa $S \setminus T = V$. Mitä suurempi opetusjoukko on suhteessa testijoukkoon, sen todennäköisemmin verkon antama luokittelu jollekin testijoukon näytteelle, jota ei käytetä opetuksessa, on oikea. Oppimisen myötä neuroverkolle voidaan laskea erilaisia tunnuslukuja. Opetusjoukon luokittelussa esiintyy *opetusvirhe* (en. learning error, arvot 0–1 tai 0–100%) ja testijoukon luokittelussa vastaavasti *testivirhe* (test error).

Kirjallisuudessa eritellään usein myös *validointijoukko* liittyen opetusalgoritmin sisäiseen optimointiin. Tässä tutkielmassa ei käytetä varsinaisesti validointijoukkoa vaan *validointiproseduuria*, joka esitellään kolmannessa luvussa.

Kun otos on tilastollisesti ottaen tasaisesti edustettu oppimistehtävässä ja opetus- ja testijoukoissa, kokonaisoppimisvirhe skaalautuu yli useiden samalla verkkorakenteella saatujen oppimistulosten. Tutkielmassa tämä varmistettiin leikkausalgoritmilla, joka valitsee otoksesta testijoukkoja, joiden luokkajakauma noudattaa otoksen luokkajakaumaa niin tiiviisti kuin mahdollista kyseisellä otoksella ja testijoukolla. Ylijäävä virhe jakautuu satunnaisesti tasajakauman mukaan kaikille luokille, jolloin toistojen lisääntyessä luokat ovat oikein edustettuina testi- ja opetusjoukoissa [J-RNS]. Menettelyllä on myös *testivirheen* varianssia laskeva vaikutus.

On tapauksia, joissa verkon oletetaan noudattavan jakaumaa tai mallia, jota ei todellisuudessa lainkaan ole olemassa tai jota (edelleen toisessa tapauksessa) ei voida rakentaa syötteenä käytettävän datan perusteella [Rissanen, 1996]. Sama oppimistehtävä voi myös olla jonkin oppimisalgoritmin kannalta ratkeamaton, mutta ratkeava toisella oppimisalgoritmilla [Valiant, 1984]. Oletuksen mukaan ongelmallinen opetustilanne johtaa huonoon oppimiseen ($< 50\%$ oikeita luokitteluja) ja lisäksi oppimistulosten variaatio on suurta. Toinen oletus on, että mitä paremmin oppimisalgoritmi soveltuu annettuun tehtävään, sitä pienempää on tulosten variaatio ja toisaalta sitä tasaisempaa oppiminen. Verkon muodostaman kuvauksen oikeellisuudesta pyritään yleensä varmistumaan tarkastelemalla verkon yleistämiskykyä testisyötteillä, kuten edellä kuvattiin. Lisävarmuutta opetustehtävän ymmärtämisessä saadaan myös analysoimalla tilastollisesti riippuvuuksia opetusdatan eri piirteiden välillä [Juhola, 2012].

MLP-verkkoja ja niiden opetusalgoritmeja käsitellään laajasti tutkimuskirjallisuudessa. Esittelen seuraavassa lyhyessä katsauksessa joitakin verkkoja koskevia lähteitä.

Minsky ja Papert [1969] osoittavat, että yksikerroksinen perceptron-verkko ei kykene epälineaarisiin erottelutehtäviin. Rumelhart ja muut [1986] osoittavat, miten useampikerroksinen MLP-verkko saattaa tehdä epälineaarisia erotteluja sekä esitte-

levät EBP-opetusalgoritmin. Sarkar [1995] esittää katsauksen EBP-algoritmiin kehitettyjen parannusten ja muunnosten päälinjoihin. Duda [2001] antaa selkeän yleisesityksen MLP-verkkojen matematiikasta ja käytöstä hahmontunnistuksessa. Anthony [2001] johtaa diskreettien neuroverkkomallien ominaisuuksia matemaattisesti opettavaisessa teoksessa. Hyvärinen ja Bingham [2003] esittävät mielenkiintoisen tulkinnan MLP-verkon yhteydestä epälineaarisiin regressiomalleihin. Hinton ja muut [2006] esittävät metodin syvien MLP-verkkojen onnistuneeseen opettamiseen. Erhan ja muut [2009] analysoivat syvien MLP-verkkojen opettamisen ongelmia ja esiopetusta. Rynkiewicz [2012] esittää lyhyen katsauksen MLP-verkkojen ylisovittumisen ongelmasta, jota käsitellään alempana. Puma-Villanueva ja muut [2012] esittävät lyhyen katsauksen MLP-verkkojen arkkitehtuurin automaattisesta optimoinnista.

Tabulaatikkohaku

Tutkielmaa varten kehitettiin uusi oppimisalgoritmi, *tabulaatikkohaku*, jota kuvataan tässä osiossa lyhyesti. Menetelmä perustuu kirjallisuudessa esiteltyyn tabuhakuun [Sexton & al., 1995, Pernía-Espinoza et al., 2005] sekä EBP-algoritmiin. Sextonin ja muiden tabuhaku generoi toistuvasti ratkaisuavaruuteen uusia satunnaisia ratkaisuja, joista muodostetaan tabulista. Ratkaisua lisättäessä sitä verrataan tabulistaan, jonka perusteella se voidaan hylätä. Tabulistan käytön lisäksi algoritmi sisältää voimistamisvaiheen, jossa parhaan löytyneen pisteen lähiympäristöstä generoidaan lisää ratkaisuja [Sexton & al., 1995]. Virhegradienttiin perustuva virhettä optimoiva EBP-algoritmi on esitelty lyhyesti ensimmäisessä luvussa (ks. [Rumelhart et al., 1986]).

Tabulaatikkohaku on stokastinen menetelmä, joka ylläpitää tabulistaa oppimisongelmaan jo löydetyistä ratkaisuista. Kukin tabu on laatikko ratkaisuavaruudessa. Uusi laatikko luodaan generoimalla ensin satunnainen ratkaisu ja kehittämällä sitä sitten syvyyshaun (EBP) avulla.

Mikäli uusi tabu osuu jonkin edellisen sisään eli generoitu piste sijaitsee jo löydetyn laatikon sisällä, se hylätään; muussa tapauksessa laatikon toinen kulma lasketaan EBP-algoritmin avulla ja saatu tabulaatikko lisätään tabulistaan. EBP-haun syvyys saadaan kaavasta $u-(k-600)/200$, missä $u \in U[0,7]$ ja k on ratkaisuavaruuden koko. EBP-algoritmissa on käytettävä kyllin pientä oppimiskerrointa tabuhaun toimivuuden takaamiseksi, tutkielmassa arvo on 0.1.

Sextonin ja muiden tabuhausta tabulaatikkohaku poikkeaa siten, että tabun muodostavaa hyperkuutiota määrittää yhden pisteen sijasta kaksi pistettä. Syvyyshaku tässä muistuttaa siis jossakin määrin Sextonin ja muiden voimistamisvaihetta,

kuitenkin sillä erotuksella, että syvyyshaku tehdään jokaisen uuden tabun luomisen yhteydessä. Generoitu piste ja johdannainen piste määrittelevät n-ulotteisessa ratkaisuavaruudessa sijaitsevan laatikon.

Algoritmi näyttää suoriutuvan erityisen hyvin tilanteessa, jossa piiloneuronien määrä ei ole kovin suuri. Tästä syystä EBP-algoritmin iteraatioita leikataan sen mukaan, mitä enemmän ratkaisuavaruudessa on piirteitä. Näin hitaasti kohti paikallista virheminimiä etenevä EBP ei ole ehtinyt kovin kauas ensimmäisestä pisteestä. Hyvin suurilla verkkorakenteilla tabulaatikkohaku tuottaa yleensä alioptimaalisen ratkaisun, josta tabuhaun jälkeen syvyyshaun tekevä EBP-algoritmi ei pääse pakenemaan. Ongelmaa voidaan säädellä pienentämällä laatikon marginaalia, joka on parametri, joka kertoo, kuinka suuren laatikon yksinäinen piste muodostaa.

Haun vahva puoli on hyvin ennustettava lyhyehkö suoritusaika, sillä odotusarvo tabulistan ja uuden pisteen vertaamisen aiheuttamista laskutoimituksista saadaan tilastollisesta sarjasta $1 \cdot (1-t) + 2 \cdot t \cdot (1-t) + 3 \cdot t^2 \cdot (1-t) + 4 \cdot t^3 \cdot (1-t) + ...$, jossa termit lähestyvät nopeasti nollaa kun t on pieni. t on tässä bernoullitodennäköisyys, jonka määrittää marginaalin koko suhteessa ratkaisuavaruuden dimension laajuuteen. Jos esimerkiksi $t = 1/10$ (laatikon marginaali on kahdeskymmenesosa arvojen vaihteluvälistä dimensiossa), tulee vertailuoperaatioiden oletusarvoksi $< 1.23 \cdot l$, missä l on tabulistan koko.

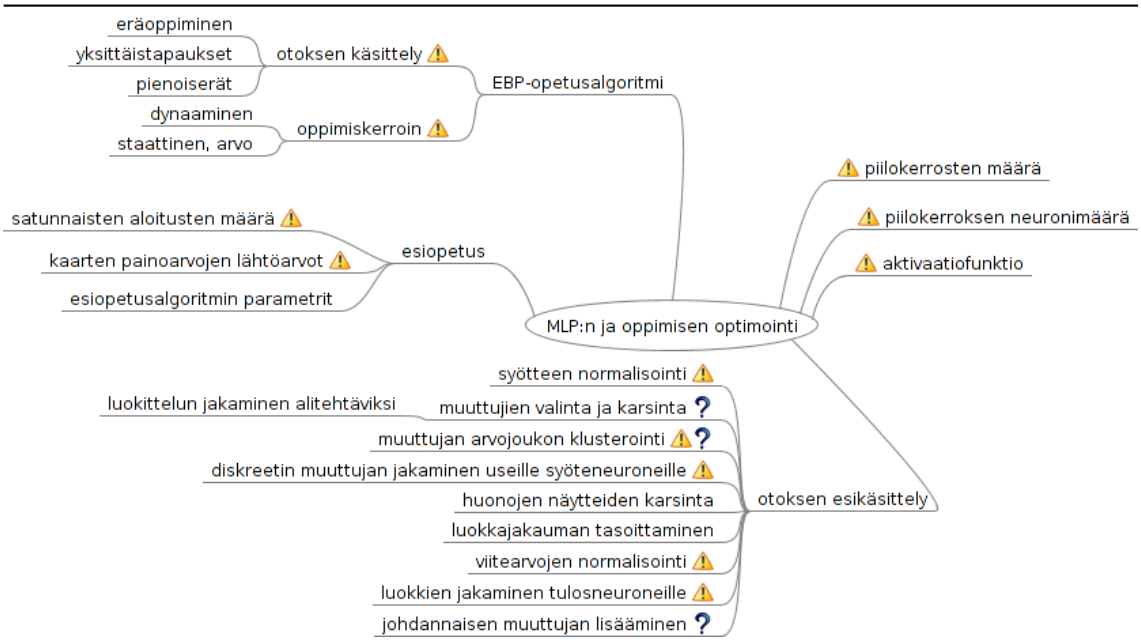
Tässä esitettyä hakualgoritmia voidaan kehittää edelleen kehittämällä heuristiikka, joka säätelee marginaalia sen perusteella, miten kaukana generoitu piste näyttää olevan paikallisesta minimistä hakuavaruuden virhepinnalla. Toisaalta tabulistan kasvaessa voidaan optimoida laatikkojen järjestystä niiden suuruuden perusteella sysäämällä suurimmat eli todennäköisimmin vertailun katkaisevat tabut listan kärkeen. Suuruusluokkaa O(1) oleva aika, joka kuluu yksinkertaisessa versiossa tabun lisäämiseen listaan, voidaan säilyttää käyttämällä osittain järjestettyä listaa, johon lisäys perustuu tilastolliseen arvioon siitä, mihin kohtaa listaa uusi laatikko kuuluu.

3 MLP-arkkitehtuurin optimointi

Edellisessä luvussa kuvattiin MLP-verkon rakenteellisia piirteitä sekä niihin ja verkon opettamiseen liittyviä ongelmia. Tämä luku käsittelee monikerroksisen verkon arkkitehtuurin optimointia tiettyä luokittelutehtävää varten.

Luokittelutehtävän ratkaisemiseksi voidaan käyttää rakenteeltaan vaihtelevia neuroverkkoja, jotka suoriutuvat luokittelusta eri tavoin. Arkkitehtuurin optimointiongelman ratkaisemiseksi voidaan valita erilaisia lähestymistapoja. Sopivat parametrit on mahdollista etsiä yrityksen ja erehdyksen kautta ilman formaaleja työkaluja – tämä on luultavasti käytetyin ja tuloksellisin tapa, kun se perustuu opetettavan ongelman ja menetelmien tuntemukseen, mutta mahdollistaa arviointivirheitä, joita systemaattiset laskennalliset menetelmät pyrkivät haarukoimaan. Etsittäessä ratkaisua ohjelmallisesti tehdään haku tiettyyn *hakuavaruuteen*, jonka ulottuvuudet vastaavat määrättyjä verkon piirteitä, kuten ensimmäisessä luvussa kuvailtiin.

Kuvassa 3.1 on assosioitu verkkoon ja oppimistehtävään liittyviä osa-alueita. Valitut hakuavaruuden piirteet ja automaattiset esikäsittelytehtävät on korostettu huomiomerkillä. Manuaalisen esikäsittelyn vaiheet on korostettu kysymysmerkillä.



Kuva 3.1 Optimoinnin osa-alueita. Käsitekartan ylemmissä haaroissa on kuvattu neuroverkon piirteiden parametrointia sekä opetustilanteen parametrointia ja alemmassa haarassa ennen opettamista tapahtuvaa otoksen esikäsittelyä.

Valittu hakuavaruus sisältää myös piilokerrosten määrän. Kuten edellä todettiin, piilokerrosten rajoittamista yhteen puoltaisi se seikka, että EBP-algoritmin optimointiongelma on sitä vaikeampi, mitä enemmän kerroksia verkossa on [Erhan et al., 2009]. Verkon piilokerrosten määrän kasvaessa myös opetettaessa käytettävän energiafunktion sisältämä virhe kasvaa sen myötä, mitä kauemmas opetusalgoritmi peräytyy tuloskerrokselta [Rumelhart et al., 1986]. Useiden piilokerrosten toteuttaminen hakuavaruuteen lisää lisäksi haun monimutkaisuutta ja kasvattaa sen vaatimaa aikaa. Puma-Villanueva ja muut [2011] ovatkin todenneet hakuavaruuden laajuuden tuovan mukanaan laskennallisen hinnan, joka voi jopa muodostua esteeksi MLP-optimointeja tehtäessä. Toisaalta neuronimäärän pitäminen piilokerrosten yli vakiona Glorot'n ja Bengion [2010] analyysin mukaisesti yksinkertaistaa jonkin verran ongelmaa. Koska useiden piilokerrosten menetelmät ovat valtavirtaa nykytutkimuksessa ja toisaalta koska alustavat testiajot antoivat lupaavia tuloksia monikerroksisten verkkojen oppimiskyvystä, otettiin parametri mukaan tähän opinnäytetyöhön. Työn vaativuutta lievennettiin vastaavasti jättämällä pois vertailu täydelliseen brute force -hakuun. Haun laajuutta ja ajallisia ulottuvuuksia kuvataan tarkemmin neljännessä luvussa.

Piiloneuronien määrä on tärkein optimoitava asia. Piilokerroksen koon alarajaksi voidaan asettaa 1. Teoreettinen yläraja on puolestaan opetettavan otoksen koko. Tällä rakenteella verkko voi antaa oikean tuloksen vielä kieroutuneessa tilanteessa, jossa jokainen otoksen näyte edustaa toisista näytteistä tilastollisesti riippumatonta luokittelua (vrt. [Hyvärinen & Bingham, 2003]). Muissa tilanteissa näin suuri neuronimäärä johtaa *ylisovittumiseen*, jota kuvattiin toisessa luvussa.

Jos oikeaa topologiaa etsitään koneellisesti, voidaan yhtäältä tehdä tyhjentävä *brute force* -haku. Tyhjentävän haun käytyä läpi koko hakuavaruuden voidaan tutkia sitä, mikä topologia antoi eniten oikeansuuntaisia tuloksia.

Yksinkertaiselle brute force -haulle on vaihtoehtoja, jotka voi karkeasti ottaen jakaa kahteen kategoriaan.

Konstruktiiviset algoritmit käyvät järjestelmällisesti läpi tutkittavaa ratkaisuavaruutta, mutta keräävät ja hyödyntävät suorituksensa optimoinnissa rakenteellista tietoa ongelmasta sekä mahdollisesti tilastollista tietoa saavuttamistaan väliratkaisuista. Esimerkkinä on Puma-Villanuevan ja muiden haku satunnaisesti yhdistettyihin rakenteeltaan MLP:tä vapaampiin eteenpäin syöttäviin monikerroksisisiin topologioihin [Puma-Villanueva et al., 2012]. Haussa Puma-Villanueva ja muut hyödyntävät useita tekniikoita lähtien neuronien dynaamisesta lisäämisestä ja poistamisesta ja oppimisvirheen kontrollin höllentämisestä neuronien keskinäisen informaation mittaamiseen. Tämän tutkielman puitteissa rakennettiin yksinkertainen konstruk-

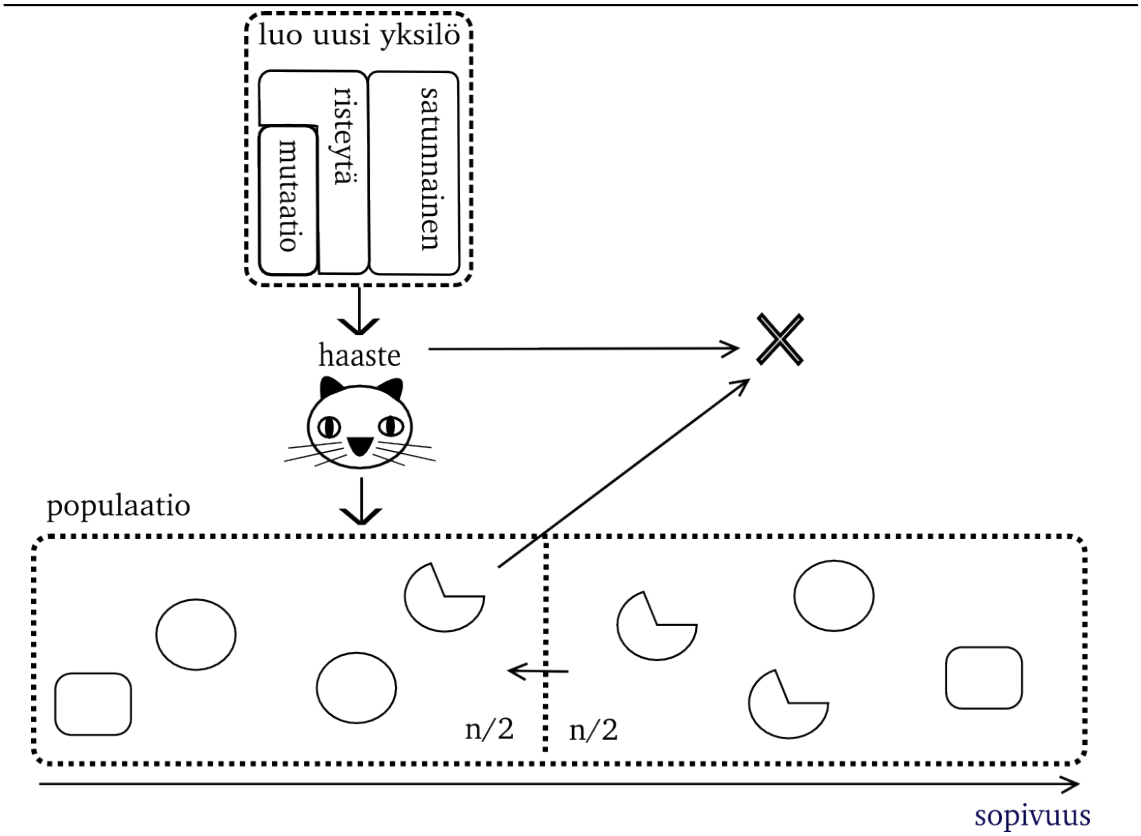
tiivinen algoritmi, joka karsii pois hedelmättömiä topologioita läheisyyssäännön perusteella. Konstruktiivisia algoritmeja ei hyödynnetä tutkielman laskennassa, mutta päätösluvussa esitetään kehitysehdotuksena eräs konstruktiivinen algoritmi, joka saattaisi suoriutua ajallisesti alla kuvattua geneettistä algoritmia paremmin.

Stokastiset hakualgoritmit tekevät ei-tyhjentävän haun samaan avaruuteen kuin brute force -hakukin, mutta suorituksen eteneminen ei ole samalla lailla järjestelmällistä kuin konstruktiivisessa algoritmissa. Stokastisen lähestymistavan hyöty perustuu siihen, että voidaan kehittää heuristiikkojen avulla hyviä ratkaisuja käymättä läpi kaikkia vaihtoehtoja, mikä puolestaan säästää resursseja. Toisaalta koska ratkaistavan ongelman ominaisuuksia (tai ongelman ratkeavuutta ylipäätään) ei välttämättä tunneta, on satunnaisuuteen perustuvan algoritmin käytössä aina riski, että sivuutettiin olennaisia rakenteita. Erilaiset *geneettiset algoritmit*, joista yhtä käytetään tässä tutkielmassa, kuuluvat tähän kategoriaan.

Stokastisen hakualgoritmin toiminta pohjautuu yritykseen ja erehdykseen sekä jonkinlaiseen *valintafunktioon*, joka mittaa algoritmin generoimien välitulosten onnistuneisuutta eli *hyvyyttä*. Algoritmi valitsee parhaana pitämänsä ratkaisun tai ratkaisut tulokseksi. *Geneettisten algoritmien* yhteydessä käytetään usein biologisesti värittynyttä termiä *sopivuus* (fitness) kuvaamaan ratkaisuperustetta. [Sivanandam & Deepa, 2008].

Käytetty geneettinen algoritmi perustuu arkkitehtuurien *populaatioon*, jonka kokoa rajoitetaan kiinteällä leikkurilla. Populaatioon syntyy uusia yksilöitä sekä täysin satunnaisesti että kaksi vanhaa yksilöä risteyttämällä. Risteytykseen kuuluu myös ajoittaisia satunnaisia mutaatioita vaihtelevissa *geeneissä*, hakuavaruuden eri dimensioiden sisällä. Populaatiota karsitaan siten, että siinä on kaiken aikaa ennalta määrätty osa sopivimpia löytyneitä arkkitehtuuri–oppimisalgoritmi-pareja, ja loput ovat satunnaisesti vähemmän sopivia. Lisäksi populaatioon pääsyä on rajoitettu karsinta-algoritmin avulla, joka esittää heikoksi uskomalleen yksilölle haasteen. Haasteessa epäonnistuminen johtaa yksilön hylkäämiseen niin, ettei se pääse osaksi populaatiota. Arkkitehtuurin sopivuus on arvo, joka ilmaisee, miten hyvin se keskimäärin suoriutuu otoksen luokittelusta, ja haastealgoritmi on toteutettu sekin MLP-neuroverkolla. Algoritmin toimintaa havainnollistetaan kuvassa 3.2.

Algoritmin päättymisehtona on laskentojen maksimäärä tai algoritmin suoritusaika, ja sen tarkemmat parametrit esitellään seuraavassa luvussa. Haun käyttämät diskreetit hakuavaruudet ovat edullisia jatkuviin nähden, sillä tyhjentävä haku on mahdollinen vain diskreetille avaruudelle, kun sen mikään dimensio ei ole ääretön. Haun suuruusluokan tietäminen auttaa myös suunnittelussa ja sitä voitaisiin mahdollisesti käyttää lisäheuristiikkojen perustana.



Kuva 3.2 MLP-arkkitehtuurien populaation säätely. Populaatio on jaettu kahteen osaan, sopivimpiin (oikealla) ja vähemmän sopiviin. Geneettinen algoritmi tuottaa uusia arkkitehtuuriyksilöitä sekä täysin satunnaisesti että risteyttämällä populaatiossa kaksi jäsentä keskenään. Populaation kokoa leikataan tasaisin väliajoin kiinteällä leikkurilla, jolloin osa yksilöistä poistetaan. Lisäoptimointina uuden yksilön pääsyä populaatioon vartioi haastealgoritmi, joka mittaa nopeasti kohdearkkitehtuurin suoriutumista tehtävässä. Algoritmi sivuuttaa historian perusteella hyviksi arvioimansa arkkitehtuurit keskittyen uusiin ja huonoiksi osoittautuneisiin yksilöihin [J-GNS].

Käyttäen useita oppimisajoja haku laskee arkkitehtuurin opetus- ja testivireen variansseineen. Näin saadaan arvio siitä, millainen tietyn rakenteen oppimistulos on pitkällä tähtäimellä.

Varsinaisen rakenteellisen haun lisäksi verkon arkkitehtuurin optimointia voidaan automatisoida myös monilta muin osin. Käytetty menetelmä sisältää eri vaiheita datan esikäsittelyssä ja verkon syöte- ja tuloskerroksen rakenteiden määrittämisessä. Varsinainen geneettinen algoritmi on osa laajempaa optimointiprosessia ja parametrointiin kuuluu manuaalinen työvaihe. Näin ollen menetelmä ei ole aivan kokonaisuudessaan automatisoitavissa ohjelmoimalla.

Menetelmä ottaa syötteenään ensimmäisessä luvussa kuvatun mukaisesti otoksen ja haku-avaruuden määrittelemän tietueen (S, A). Hakua ei tässä tutkielmassa sovelleta dynaamisesti laajeneviin otoksiin (reaaliaikainen oppiminen).

1. esikäsittele otos

    - karsi pois huonot muuttujat (manuaalinen analyysi perustuen aineiston tarkasteluun ja kohdealueen tuntemukseen)
    - klusteroi suurta heiluntaa sisältävät muuttujat kuten potilaan ikä (manuaalinen vaihe tämän työn piirissä, voidaan automatisoida tilastollisin menetelmin)
    - normalisoi syötemuuttujat
    - pilko ei-numeeriset syötemuuttujat useaksi binäärimuuttujaksi
    - pilko pienen arvojoukon syötemuuttujat useaksi binäärimuuttujaksi
    - täydennä puuttuvat arvot satunnaisesti muuttujan jakaumasta (automaattinen vaihe, ei käytetty tutkielmassa, sillä aineistoissa ei puuttuvia arvoja)
    - luokittelutehtävässä tasoita tarvittaessa luokkien jakauma kasvattamalla otosta duplikaattitietueilla (manuaalinen vaihe tämän työn piirissä, ei sovellettu yhteenkään koeaineistoon)
    - luokittelutehtävässä pilko kukin tulosmuuttuja sen luokkia vastaavasti useaksi muuttujaksi

2. luo uusi yksilö populaatioon (leikkaa tarvittaessa populaation kokoa)

3. alusta uusi verkko satunnaisluvuilla

4. valitse opetusjoukko ja testijoukko

5. opeta

6. testaa laskemalla verkon keskivirhe sekä opetusjoukon että testijoukon yli ja tallenna tulokset; toista palaamalla kohtaan 3 kunnes opetettu riittävän monta kierrosta, jolloin siirry seuraavaan kohtaan

7. tarkista laskennan määrä- ja aikaehdot. Jos valmista, algoritmi päättyy, muuten kohtaan kaksi.

Luokittelu syöte- ja luokkamuuttujiin perustuu tutkielmassa aineiston ulkopuolisiin tietoihin (aineiston liitteet). Kaikkien muuttujien normalisoinnissa käytetään ns. *minmax-normalisoijaa*, jonka normalisoima arvo $a' \in [0, 1]$:

$$a' = \frac{a - min(A)}{max(A) - min(A)}, \tag{3.1}$$

missä a on alkuperäinen (mitattu) arvo ja A normalisoitavan muuttujan arvojoukko otoksessa. Muuttujan keskiarvoa ja keskihajontaa käyttävä standardointi

$$a' = \frac{a - \overline{A}}{\sigma_A} \tag{3.2}$$

suoriutui alustavien testien perusteella keskimäärin minmax-normalisoijaa huonommin.

Manuaaliseen analyysiin perustuen pois karsitaan joitakin mittausaineistoon kuulumattomia muuttujia, esimerkiksi tietokannan hallintaan liittyviä parametreja.

Mikäli jokin muuttuja saa vain muutamia arvoja, sen syöte jaetaan luokittelun parantamiseksi automaattisesti usealle syötemuuttujalle. Pilkottavaksi valitaan muuttujat, jotka saavat alfanumeerisia arvoja sekä kokonaislukuarvoja saavat muuttujat, joiden arvojoukko on kapea. Myös muuttujat, jotka saavat vain muutamia reaalisia arvoja, esimerkiksi joukosta $\{0.5, 1.0\}$, pilkotaan. Luokkamuuttujat pilkotaan aina, eli ne jaetaan automaattisesti useaksi tulosneuroniksi.

Syötemuuttujien automaattinen pilkkominen on yleisessä tapauksessa ongelmallista, sillä muuttujien määrän ja otoksen koon välillä on korrelaatioita, jotka vaikuttavat oppimiseen, kuten toisessa luvussa kuvattiin. Tutkielman lähdekoodissa jakamisen kynnys on parametroitu erillisellä metaparametrilla, jonka oletusarvona käytetään kahtakymmentä erillistä arvoa. Tätä laajempia muuttujia ei pilkota [J-SMP].

Joissakin tapauksissa verkon luokittelukykyä voidaan hieman parantaa lisäämällä syötteeseen johdannaisia muuttujia.

Yksittäinen useista epokeista koostuva opetussessio rakentuu seuraavasti. Aluksi luodaan uusi neuroverkko, jota opetetaan, kunnes oppimistulos ei enää parannu tai on saavutettu ennalta valittu keskineliövirheen (MSE) alaraja. Käyttäen verkon sopivuuden suhteen ahnetta oppimisprosessia kirjataan senhetkinen verkko ja sen antama luokittelu tulokseksi, jos se paransi edellistä sopivuustulosta.

Arkkitehtuurin sopivuus fit oppimistehtävässä määritellään seuraavasti:

$$fit = (V_T + T_T) \cdot [1 - (max(0.1, |V_T - T_T|)], \tag{3.3}$$

missä V_T on oikein luokiteltujen testisyötteiden suhde ja T_T oikein luokiteltujen opetusjoukon syötteiden suhde. Funktio optimoi rakenteen sopivuutta siten, että sekä testijoukon että opetusjoukon hyvä luokittelutulos palkitaan.

Alustavien testien perusteella yksittäinen oppimistilanne saattoi kestää puolesta sekunnista useisiin kymmeniin minuutteihin. Geneettisen hakualgoritmin kestoa rajoitettiin kuuteentoista tuntiin per optimointitehtävä ja lisäksi geneettinen algoritmi valitsi piiloneuronien määrän pieniä neuronimääriä suosivasta jakaumasta $(U[0,1])^3 \cdot (n-1) + 1$, jonka arvojoukko on sama kuin diskreetin tasajakauman $U[1, 2, ..., n]$. Opetustulokset kirjattiin tietokantaan, josta ne olivat reaaliajassa raportoitavissa.

Tulosten validoimiseksi ajettiin brute force -haku hakuavaruuden merkitsevässä osassa ja geneettisen algoritmin tuloksia verrattiin saatuihin tuloksiin.

4 Arkkitehtuurin optimoinnin tarkastelu

Kolmannessa luvussa esiteltyä optimointialgoritmia testattiin lääketieteellisillä datajoukoilla sekä generoiduilla datajoukoilla, jotka on listattu taulukossa 4.1.

Lyhenne	Datajoukko	Tyyppi	Koko	Syötemuuttujia	Luokkia
Bupa	BUPA liver disorders	Lääket.	345	6	2
Ecoli	Protein Localization Sites	Lääket.	336	7	8
Haberman	Haberman's Survival Data	Lääket.	306	3	2
New-thyroid	Thyroid gland data	Lääket.	215	5	3
Pima-indians	Pima Indians Diabetes Database	Lääket.	768	8	2
Parabola	$y = x^2 + R$	Gener.	100	3	2
Superimposed sines-2	$y = sin(x \cdot 27) + sin((x+1) \cdot 11) + R$	Gener.	1000	3	2

Taulukko 4.1: Tutkimuksessa käytetyt datajoukot. $R = sg \cdot t \cdot n$, missä sg on joko -1 tai 1, $t \in U[-5, 5]$ ja $n \in N(\mu = 0, \sigma^2 = 4)$.

Oppimistehtäväkohtainen hakuavaruus määriteltiin taulukon 4.2 mukaisesti. Hakuavaruuden laajuus eri oppimistehtävissä on listattu taulukossa 4.3. Avaruuden koko oli datajoukkokohtainen ja riippuvainen opetettavasta otoksesta, esimerkiksi $345/6 \cdot 5 \cdot 6 \cdot 3 \cdot 16 = 496800$ Bupa-datalle. Luku antaa maksimimäärän tarkasteltaville verkkorakenne–oppimisalgoritmi-yhdistelmille. Piilokerrosten rakenne oli hakuavaruuden ainoa täysin välttämätön piirre. Kokeen tulokset on listattu taulukossa 4.5 ja vertailu rajoitettuun brute force -hakuun taulukossa 4.7.

Seuraavaksi eritellään tarkemmin koeasetelmaa ensin yleisellä tasolla koskien kaikkia oppimistehtäviä yhteisesti. Tehtävät käydään myöhemmin yksitellen läpi kiinnittäen huomiota esikäsittelyyn, geneettisen haun tuloksiin ja oppimisongelmaan.

Hakuavaruutta [J-MSS] käsitteli hakualgoritmi [J-GNS]. Piilokerrosten ja niiden neuronien suhteen otettiin käyttöön seuraavat rajoitukset. Jos piilokerroksia oli useampia, kullakin oli sama määrä neuroneita [Glorot & Bengio, 2010]. Toiseksi piiloneuronien kokonaismäärää rajoitettiin niin, että jos se ylitti määrätyn ylärajan, karsittiin tarvittaessa piilokerrosten määrää. Menettelyä tukee havainto, että useampikerroksinen rakenne voi tarvita huomattavasti vähemmän neuroneita saman tehtävän luokittelemiseksi kuin kolmikerroksinen [emt.]. Lähtöoletuksena oli myös, että kukin otos sisältää jonkin verran korrelaatiota, jolloin luokitteleva malli ei koskaan tarvitse otoksen kokoa vastaavaa määrää piiloneuroneita. Piiloneuronien määrälle valittiin yläraja $n/6$ suhteessa otoksen kokoon. Yläraja haluttiin mahdollisimman ahtaaksi, sillä mitä suurempi neuroverkkorakenne, sen hitaampaa sen opettaminen

Dimensio	Arvojoukko	Koko
Piiloneuronien määrä	{1, 2, ..., n/6}	kuudennes otoksen koosta
Piilokerroksia	{0, 1, 2, 3, 4}	5*
Aktivaatiofunktio	hyperbolinen	6
	softsign	
	sigmoidinen, jyrkkyys:	
	{ 0.4, 1.0, 3.0, 9.0 }	
Esiopetus	tabulaatikkohaku, iteraatiot:	3
	{ 0, 100, 1000 }	
Oppimisalgoritmi	EBP, minibatch (10), oppimiskerroin 0.001	16
	EBP, sopeutuva, oppimiskerroin 0.001	
	EBP, minibatch, dynaaminen, alkaa arvosta 0.001	
	EBP, sopeutuva, dynaaminen, alkaa arvosta 0.001	
	EBP, minibatch, oppimiskerroin 0.01	
	EBP, sopeutuva, oppimiskerroin 0.01	
	EBP, minibatch, dynaaminen, alkaa arvosta 0.01	
	EBP, sopeutuva, dynaaminen, alkaa arvosta 0.01	
	EBP, minibatch, oppimiskerroin 0.1	
	EBP, sopeutuva, oppimiskerroin 0.1	
	EBP, minibatch, dynaaminen, alkaa arvosta 0.1	
	EBP, sopeutuva, dynaaminen, alkaa arvosta 0.1	
	elastinen EBP, minibatch (10)	
	elastinen EBP, sopeutuva	
	momentti-EBP, minibatch (10)	
	momentti-EBP, sopeutuva	

Taulukko 4.2: Hakuavaruuden dimensiot

ja suorituskyvyn arviointi on. Oppimistehtävien suuren määrän nojalla täydellinen vertailu brute force -hakuun hylättiin.

Aktivaatiofunktio otettiin mukaan hakuavaruuteen, koska funktion piirteet voisivat vaikuttaa oppimistuloksiin ja verkon ilmaisuvoimaan. Kaikki hakuun sisältyvät funktiot olivat epälineaarisia [J-SIG,J-SSG,J-TNH]. Vain joitakin mahdollisista sigmoidisista funktiosta sisällytettiin hakuun.

Lisäksi vaihdeltiin esiopetukseen sisältyvää satunnaisten aloitusten määrää sekä käytettyä oppimisalgoritmia, joilla molemmilla voisi olla positiivista vaikutusta oppimistuloksiin. Aktivaatiofunktion ja oppimisalgoritmin vaihtelulla sekä esiopetuksella kasvatettiin haun mahdollisuuksia löytää onnistunut ratkaisu. Esiopetuksessa käytettiin useiden aloitusten menetelmänä tabulaatikkohakua [J-TAB], joka on esitelty luvussa 2.

Esiopetuksen jälkeen verkkoa kehitettiin jollakin EBP-algoritmin [J-EBP] muunnoksella, jotka on listattu taulukossa 4.2. Elastinen EBP-algoritmi [J-ELA] perustuu neuronikohtaisiin oppimiskertoimiin ja eräajo-oppimiseen ja momentti-EBP [J-MOM] perustuu oppimisprosessissa keräytyvään momenttiin, joka määrittää oppimisen suuntaa taannehtivasti edellisten oppimisvirheiden perusteella [Sarkar, 1995].

Datajoukko	Hakuavaruuden laajuus S	BF, oppimistehtäviä yhteensä $(20 \cdot S)$	Gen. haussa tutkittuja	Kesto	Oppimiskesto ka
Bupa	496800	9936000	171	32h	28.1s
Ecoli	483840	9676800	487	32h	7.1s
Haberman	440640	8812800	5020	5h*	3.0s
New-thyroid	309600	6192000	247	180min*	1.6s
Pima-indians	1105920	22118400	95	32h	35.7s
Parabola	144000	2880000	505	126min*	0.5s
Superimposed-sines-2	1440000	28800000	50	32h	102.0s

Taulukko 4.3: Hakuavaruuden laajuus. Tähdellä merkityt ajot keskeytetty manuaalisesti, koska tulos ei enää parantunut.

Parametri	Arvo	Huomioita
Populaation koko	20	
Vahvojen osuus populaatiosta	50%	Populaatiota supistettaessa säilytettiin 50% sopivimpia.
P(Satunnaisen yksilön syntyminen)	0.5	
P(Risteytyminen)	0.5	
P(Satunnainen mutaatio)	0.1	Satunnainen mutaatio syntyi risteytyksen yhteydessä todennäköisyydellä 0.1 ja johti oppimisalgoritmin, aktivaatiofunktion sekä satunnaisten aloitusten arvontaan.

Taulukko 4.4: Geneettisen haun parametrit

Yksittäinen arkkitehtuuri arvioitiin seuraavasti. Datajoukko [J-DS] jaettiin opetusjoukkoon ja pienempään testijoukkoon [J-TS], joiden suhdeluvuksi valittiin kiinteä 9/1. Monimutkaisempi proseduuri hylättiin liian vaativana, sillä se olisi vaatinut syvällistä esiymmärrystä syötteiden ja luokkajakauman keskinäisistä riippuvuuksista. Opettaminen toistettiin samalla rakenteella aina 20 kertaa. Jokaisella toistolla verkon antama tulos testattiin testijoukolla ja opetusjoukolla. Tulos yksittäisiltä ajoilta sekä ajojen keskiarvoinen tulos variaatioineen kirjattiin tietokantaan [J-TRR]. Geneettisen haun tulosta verrattiin osittaiseen brute force -hakuun hakuavaruudessa.

Geneettiselle valinta-algoritmille valittiin taulukossa 4.4 kuvatut parametrit.

Hakualgoritmi [J-GNS] järjesti arkkitehtuurit ja oppimisalgoritmit luokittelukyvyn perusteella paremmuusjärjestykseen, suosien tasaveroisten oppimistulosten tapauksessa mahdollisimman pieniä verkkorakenteita ja nopeita opetussyklejä. Näin algoritmi tuotti sekä ehdotuksen opetustehtävälle käytettävästä verkon topologiasta että oppimisalgoritmista, joka soveltuisi parhaiten kyseisen verkon opettamiseen pitkällä tähtäimellä. Geneettisen haun löytämät parhaat rakenteet on esitetty tau-

Otos		Akt.f	Op.alg	Piilon	Esiop	Testivirhe % (σ^2)	Opetusvirhe % (σ^2)
bupa	P	Sigmoid 3.0	Elast 0.1	84x3	100	19.5±10.9	7.1±62.9
bupa	P	Sigmoid 3.0	Elast 0.1	42x3	1000	20.7±5.1	9.3±46.7
bupa	L	Sigmoid 1.0	Elast-MINB 0.1	0	100	32.0±12.6	40.9±56.0
bupa	L	Softsign	Elast-MINB 0.1	0	1000	38.6±23.6	41.7±36.4
	Ka			22x1	318.7	29.1	22.7
ecoli	P	Sigmoid 1.0	Moment-MINB 0.1	16x1	0.0	8.7±8.7	8.6±0.6
ecoli	P	Sigmoid 1.0	Moment-MINB 0.1	12x2	1000	9.7±8.0	8.8±1.5
ecoli	L	Sigmoid 9.0	EBP 0.1	0	0.0	13.6±9.5	13.9±0.6
ecoli	L	Sigmoid 1.0	EBP 0.1	0	1000	14.0±8.5	13.8±1.7
	Ka			12x1	350.3	20.7	19.1
haberman	P	Sigmoid 0.4	EBP 0.1	22x3	1000	25.5±4.7	15.2±14.2
haberman	P	Sigmoid 1.0	EBP-MINB 0.01	11x4	100	26.5±10.4	15.6±10.9
haberman	L	Sigmoid 1.0	EBP-MINB 0.001	0	100	26.5±8.8	21.5±4.1
haberman	L	Sigmoid 1.0	EBP-MINB 0.001	0	1000	27.5±9.6	21.6±7.0
	Ka			8x1	407.2	34.8	21.0
new-thyroid	P	Sigmoid 0.4	EBP-MINB 0.1	8x4	1000	0.7±1.0	1.9±0.2
new-thyroid	P	Tanh	EBP 0.1	18x4	100	1.6±1.5	0.4±0.4
new-thyroid	L	Sigmoid 9.0	Elast 0.1	0	1000	6.6±5.9	14.4±127.5
new-thyroid	L	Sigmoid 9.0	EBP 0.01	0	1000	8.3±5.1	6.2±2.1
	Ka			8x1	472.8	14.4	6.6
pima	P	Sigmoid 3.0	Moment-MINB 0.1	27x1	1000	21.4±7.3	12.0±37.8
pima	P	Sigmoid 3.0	EBP-MINB 0.1	37x1	1000	21.6±5.5	12.0±43.1
pima	L	Sigmoid 3.0	EBP-MINB 0.1	0	0.0	22.9±11.7	18.6±1.4
pima	L	Sigmoid 3.0	Elast-MINB 0.1	0	1000	23.1±21.6	29.8±31.4
	Ka			22x1	581.0	26.3	17.6
parabola	P	Tanh	EBP 0.1	7x2	100	0.0±0.0	0.0±0.0
parabola	P	Sigmoid 0.4	EBP 0.1	11x1	0.0	0.0±0.0	0.0±0.0
parabola	L	Tanh	EBP-MINB 0.1	0	0.0	10.5±4.4	14.6±3.1
parabola	L	Sigmoid 3.0	Moment-MINB 0.1	0	0.0	11.5±6.2	13.9±2.3
	Ka			1x1	354.6	12.4	12.4
sup-2-sines	P	Sigmoid 3.0	Elast 0.1	57x2	100	24.7±14.8	20.0±98.9
sup-2-sines	P	Sigmoid 3.0	Elast 0.1	37x2	100	30.1±24.6	26.9±117.3
	Ka			20x2	396.0	40.8	39.3

Taulukko 4.5: Tulokset opetustehtävittäin. Toisessa sarakkeessa merkitty, onko rivillä haun P(aras), L(ineaarinen) vai Keskiarvoinen tulos. Oppimisalgoritmissa EBP 0.1 tarkoittaa EBP-algoritmin yksittäistapausoppimista oppimiskertoimella 0.1, MINB minibatch-oppimista, Elast elastista EBP-algoritmia ja Moment momenttitermi-EBP:tä.

lukossa 4.5.

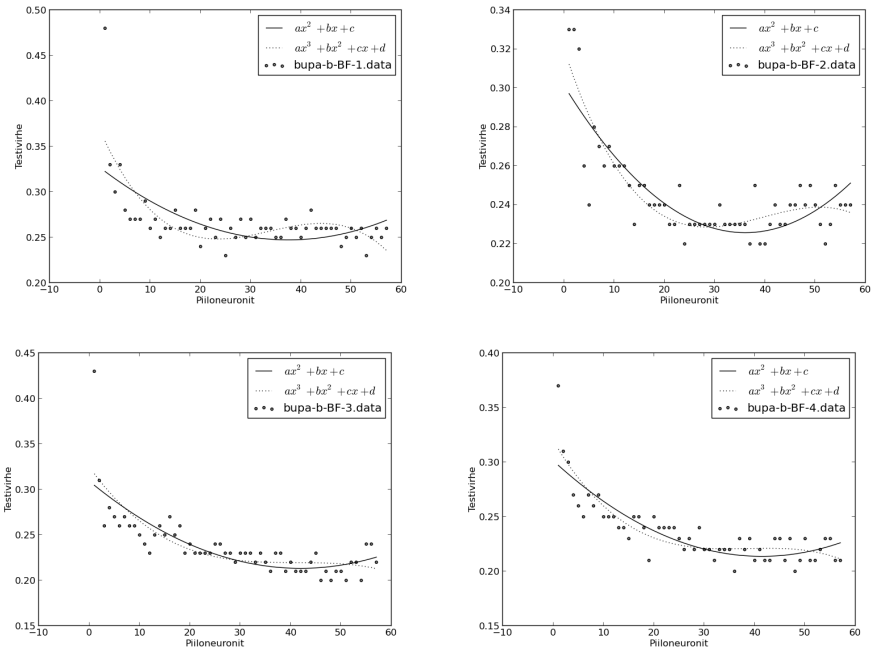
Geneettisen algoritmin ajamisen jälkeen tutkittiin kymmentä parasta haun tuottamaa tulosta ja niiden perusteella määritettiin hakuavaruus validoinnissa käytettävälle brute force -haulle taulukon 4.6 mukaisesti. Haun kooksi tuli $\frac{4}{6}n$, missä n on otoksen koko. Taulukossa 4.7 ja kuvissa 4.1 – 4.7 on eritelty brute force -haun ja geneettisen haun yhtenevyyksiä. Geneettisen algoritmin tuloksia eri oppimistehtävissä on tarkasteltu seuraavissa alaluvuissa.

Dimensio	Arvojoukko	Koko
Piiloneuronien määrä	{1, 2, . . . , n/6}	kuudennes otoksen koosta
Piilokerroksia	{1, 2, 3, 4}	4
Aktivaatiofunktio	mediaani kymmenen gen. parhaan joukosta	1
Aktivaatiofunktio, jyrkkyys	„	1
Esiopetus	„	1
Oppimisalgoritmi, algoritmi	„	1
Oppimisalgoritmi, algoritmi	„	1
Oppimisalgoritmi, oppimiskerroin	„	1
Oppimisalgoritmi, dynaamisuus	„	1

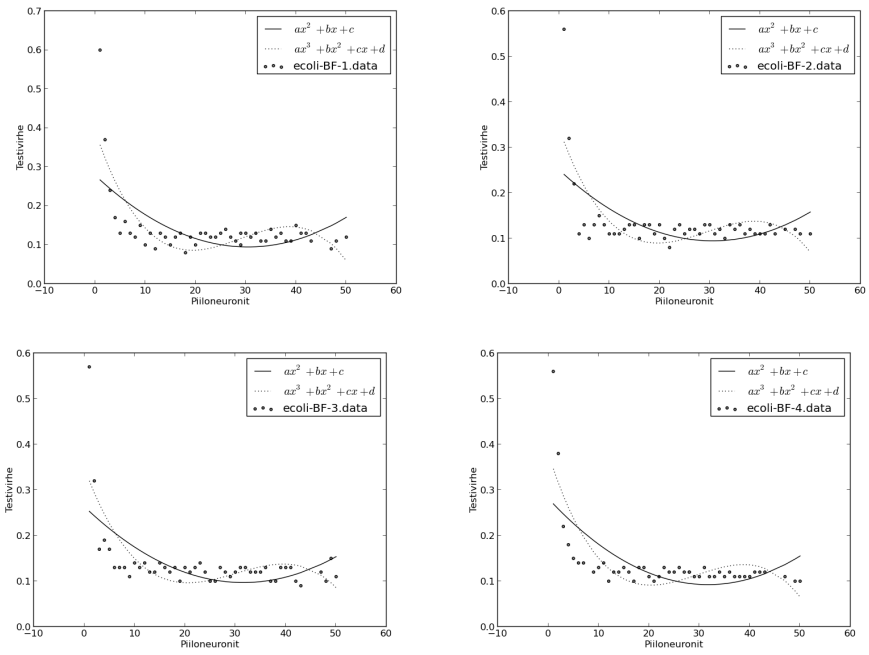
Taulukko 4.6: Validoinnissa käytetty haku (brute force).

Otos	Suosittelija	P-1	P-2	P-3	P-4
Bupa	BF-2	37.4	36.1	42.0	41.4
	BF-3	23.5	28.4	30.3	34.5
	Gen	49.0	33.7	54.3	42.7
Ecoli	BF-2	30.4	30.5	31.6	31.7
	BF-3	19.4	19.6	20.6	20.4
	Gen	16.0	16.3	16.3	8.7
Haberman	BF-2	39.4	30.7	34.9	35.1
	BF-3	29.1	20.1	22.8	24.2
	Gen	9.3	12.3	16.3	10.7
New-thyroid	BF-2	22.2	21.6	21.9	21.9
	BF-3	13.8	13.8	14.0	14.0
	Gen	11.7	12.7	10.3	15.7
Pima-indians	BF-2	78.6			
	BF-3	48.1			
	Gen	31.3			
Parabola	BF-2	10.4	10.4	10.4	10.7
	BF-3	6.7	6.9	6.8	7.3
	Gen	6.7	4.3	4.3	5.0
Sup.sines-2	BF-2		100.3		
	BF-3		95.9		
	Gen		57.0		

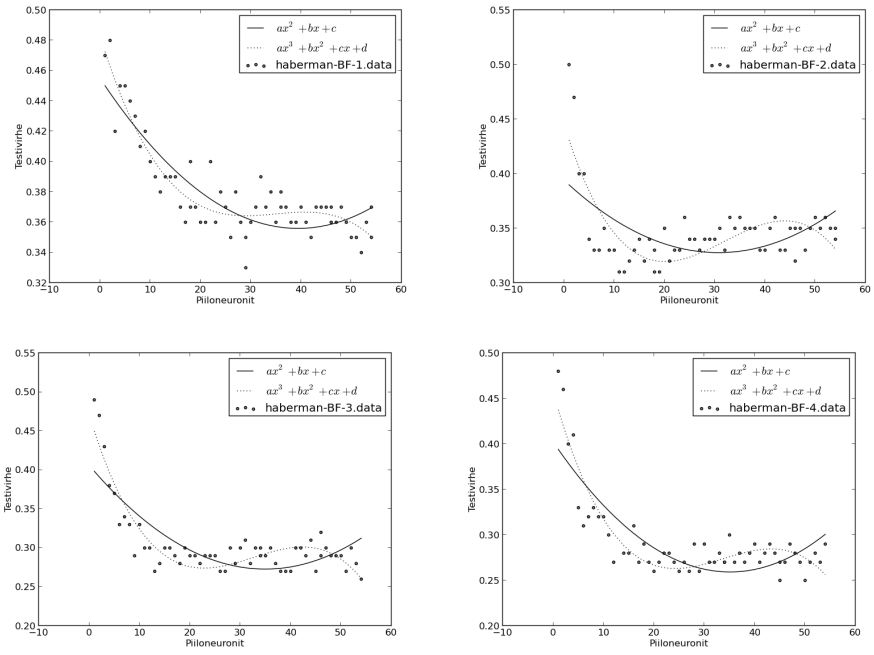
Taulukko 4.7: Geneettisen algoritmin suosittelemat verkkorakenteet. Vertailu brute force -hakuun. BF-haku ajettiin koko topologiajoukolle mediaanisella oppimisalgoritmilla, aktivaatiofunktiolla ja esiopetusparametrilla. Taulukossa on esitetty haun perusteella ehdotus piiloneuronimäärästä. Sarake P-1 merkitsee yhtä piilokerrosta, P-2 kahta piilokerrosta ja niin edelleen. BF-2-rivi sisältää brute force -haun tuloksiin sovitetun toisen asteen polynomin minimikohdan ja BF-3-rivi kolmannen asteen polynomin paikallisen minimikohdan. Gen-rivi sisältää geneettisen algoritmin löytämän testivirheeltään parhaan kolmen verkon keskiarvoisen piiloneuronimäärän. Esimerkiksi yhden piiloneuronikerroksen bupa-tehtävissä brute force -hakuun sovitettu toisen asteen polynomi ehdottaa piilokerroksen kokoa 37 ja kolmannen asteen polynomi vastaavasti kokoa 24, kun taas geneettisen algoritmin ehdottama koko on 49. Opetussyklin hitauden takia Pima- ja Superimposed sines-2 -tehtävissä ajettiin brute force -haku vain yhden ja kahden piilokerroksen verkoille.



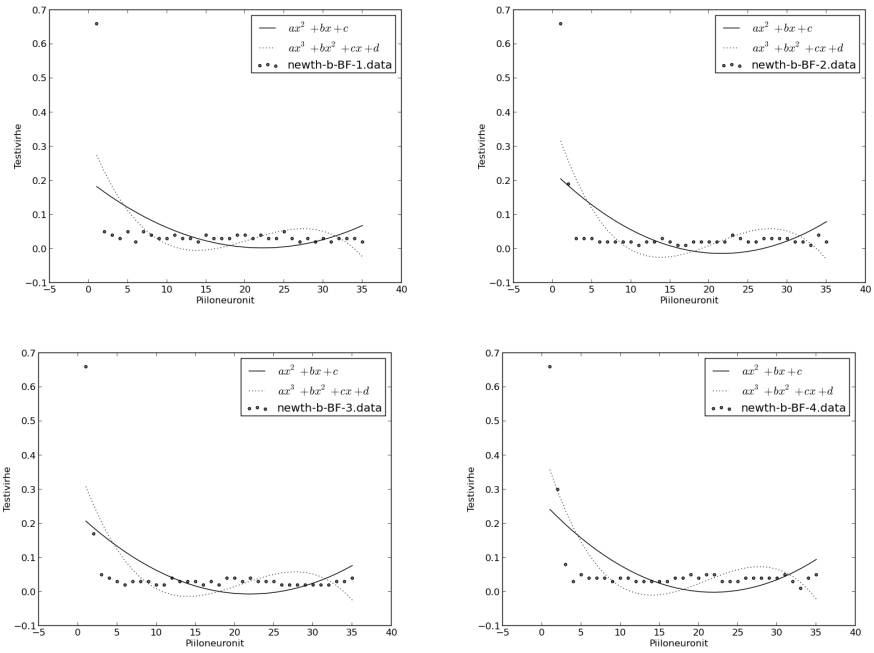
Kuva 4.1 Testivirhe osittaisessa brute force -haussa (bupa) neuronimäärän funktiona. Ylhäällä vasemmalla yhden piilokerroksen verkon testivirhe, ylhäällä oikealla kahden, ja niin edelleen. Mustalla viivalla on piirretty joukkoon sovitettu toisen asteen polynomi, jonka minimi osoittaa idealisoitua kohdetta geneettisen haun tulokselle. Todellisuudessa datajoukon muoto ei useista syistä ole täysin symmetrinen, ja toisen asteen polynomin antama tulos voi olla virheellinen. Katkoviivalla on piirretty kolmannen asteen polynomi, jonka käyttäminen sovituksessa on sitä suositeltavampaa, mitä pienempi käyrän paikallinen minimiarvo on toisen asteen polynomin minimiin verrattuna.



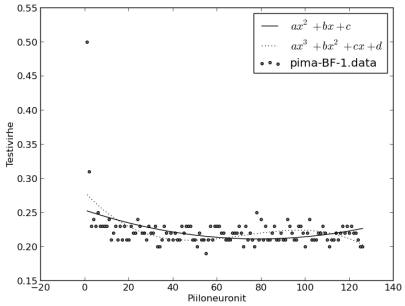
Kuva 4.2 Näytteet osittaisen brute force -haun tuloksista (ecoli). Kolmannen asteen polynomin antama ennuste on parempi kuin toisen asteen polynomin (vrt. minimien syvyys sekä näytteen jakauman muoto). Viidennen asteen polynomin antama ennuste olisi vielä tätäkin luotettavampi siirtäessään ennustetta yhä kohti geneettisen haunkin löytämää varhaista minimiä.



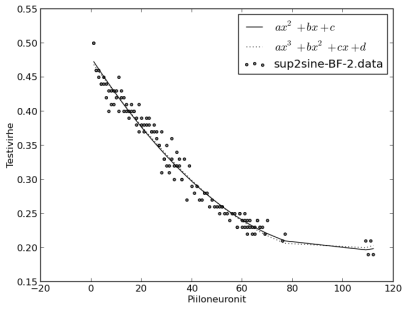
Kuva 4.3 Näytteet osittaisen brute force -haun tuloksista (haberman). Sekä hajonta että testivirhe ovat pienimpiä kolmi- ja nelipiilokerroksisissa verkoissa, joita voidaan suositella tämän perusteella. Geneettisen haun löytämät parhaat verkot ovat samoin kolmi- ja nelikerroksisia sekä neuronimäärältään yllättävänkin lähellä brute force -haun osoittamaa minimiä kolmannen asteen polynomista (vrt. alaluku 4.6).



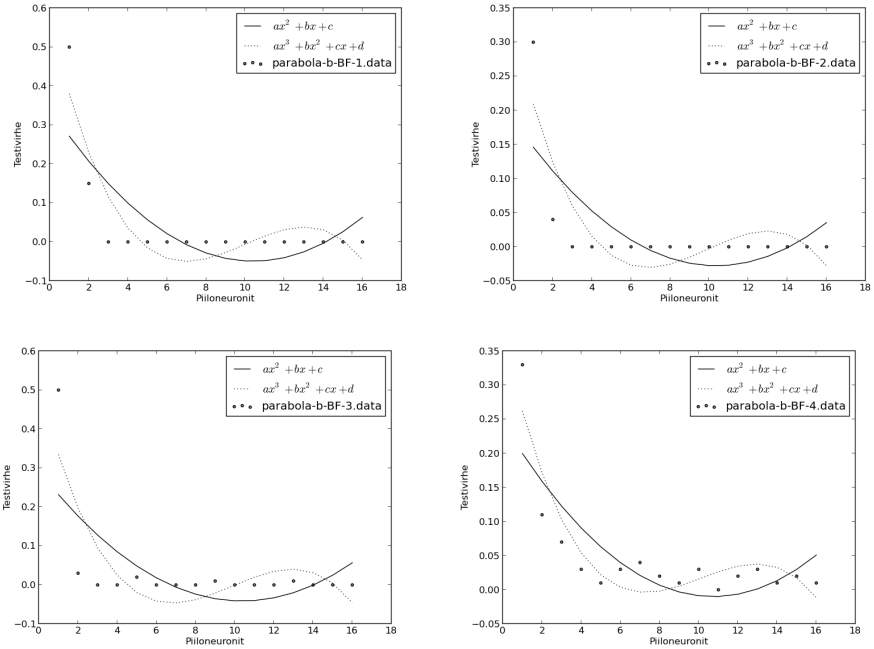
Kuva 4.4 Näytteet osittaisen brute force -haun tuloksista (new-thyroid). Hyvän opittavuuden takia näytteiden jakauma on miltei suoralla viivalla. Tämän takia mikään matalan asteen polynomi ei anna oikeaa suositusta mahdollisimman pienestä neuronimäärästä. Geneettinen algoritmi ehdottikin brute force -sovitusta pienempää piiloneuronimäärää.



Kuva 4.5 Näytteet osittaisen brute force -haun tuloksista (pima). Hajonta on suhteellisen suurta.



Kuva 4.6 Näytteet osittaisen brute force -haun tuloksista (superimposed sines). Aineistosta puuttu kaistale väleillä 66–67 ja 80–109.



Kuva 4.7 Näytteet osittaisen brute force -haun tuloksista (parabola). Otos noudattaa pian miltei suoraa viivaa, joten sovitetut polynomit eivät tässä osoita mitään erityistä minimikohtaa. Geneettisen algoritmin antama suositus on parempi, sillä pienempi rakenne on edullisempi.

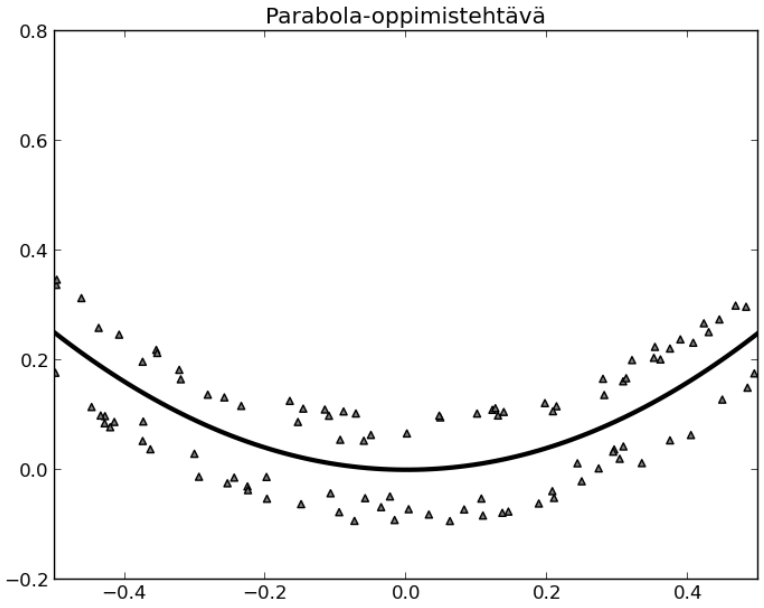
4.1 Parabola

Parabola-tehtävällä testattiin lineaarisia ja epälineaarisia erotteluja sekä oppimisalgoritmien suoriutumista verkon kaarten optimoinnista. Tehtävä on visualisoitu

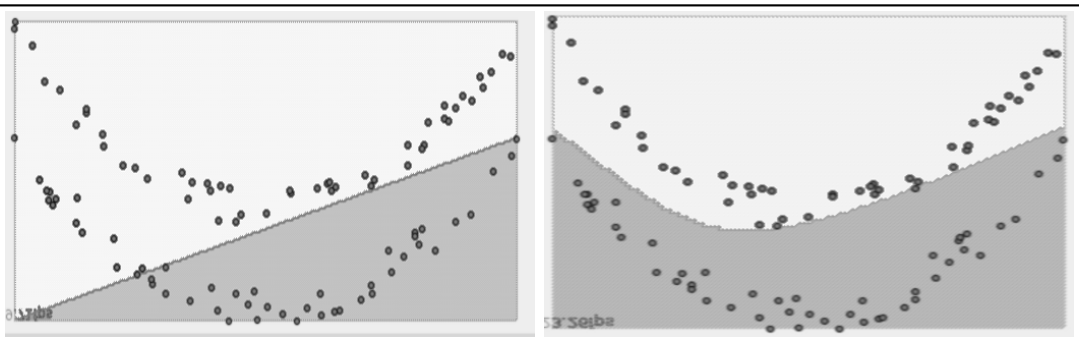
kuvassa 4.8.

Monet algoritmin valitsemat verkkorakenteet luokittelivat opetus- ja testijoukon täydellisesti ilman virheitä. Myös lineaarinen rakenne pääsi usein hyvään luokitteluun kuvan 4.9 osoittamalla tavalla, parhaimmillaan 1.5%:in keskimääräiseen testivirheeseen ja 14%:in keskimääräiseen opetusvirheeseen. Toisaalta verkot, joissa piilokerroksella oli vain yksi neuroni, jäivät nollaoppimistulokseen (50.0%, 50.0%). Monet kolmi- ja useampikerroksiset verkot, joiden piilokerroksilla oli kaksi neuronia. Yksittäisen piiloneuronin huonoa oppimistulosta voidaan selittää sillä, että se litistää lävitseen kulkevan signaalin tasaiseksi, jolloin syötemuuttujien piirteitä on mahdoton kuljettaa verkon läpi.

Sekä opetusjoukko että verkkorakenteet olivat pieniä. Haku katkaistiin 500 syklin jälkeen (126min). Kuvasta 4.7 ilmenee, että luokitteluvirhe on tasaisen olematon tietyn piiloneuronimäärän ylityttyä. Neljän piilokerroksen verkossa ovat oppimistulokset kuitenkin jo vähemmän tasaisia.



Kuva 4.8 Parabola-oppimistehtävä. Tehtävä on verrattain harva näytteiltään ja yksinkertaisuutensa vuoksi oletuksen mukaan opittavissa hyvin kahdella piiloneuronilla. Luokiteltavat arvot tulevat normaalijakaumasta $N(t, \mu, \sigma^2)$ käyrän molemmin puolin parametreilla t, $\mu = 0$, $\sigma^2 = 4$, missä t tulee tasajakaumasta $U[-5, 5]$.



Kuva 4.9 Vasemmalla lineaarinen luokittelu Parabola-oppimistehtävästä (kokonaisvirhe 15.5%). Oikealla 2x4-rakenteen antama täydellinen luokittelu.

4.2 Superimposed-sines-2

Superimposed-sines-2-tehtävällä (kuva 4.10) tutkittiin verkon suoriutumista monimutkaisesta syötepinnasta, jossa luokkien välimatka toisistaan on pieni. 1000 näytteen otos on tutkielman suurin. Ensimmäinen kuudentoista tunnin ajo ehti tutkia vain kahdeksan suurehkoa verkkorakennetta, joten sitä jatkettiin toisella ajolla (yhteensä 50 rakennetta).

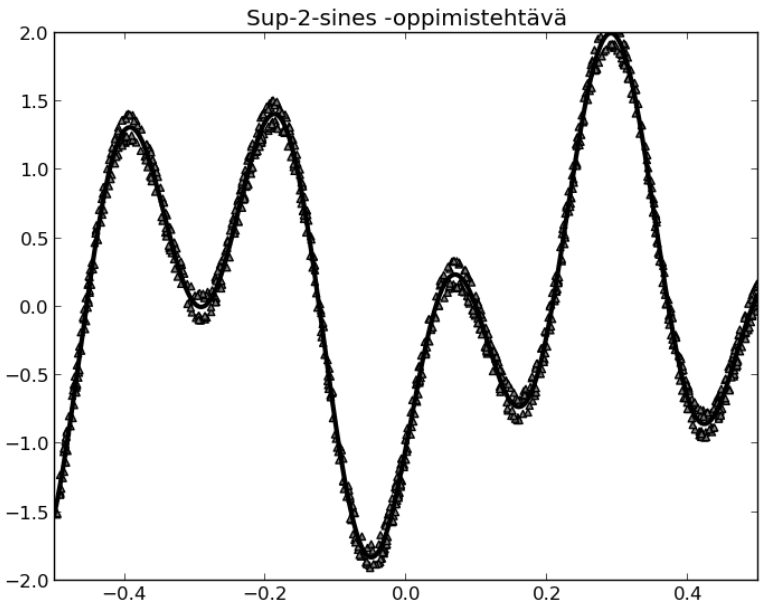
Paras luokittelija oli sigmoid(3) 57x2-verkko 20.1%:n opetusvirheellä ja 24.8%:n testivirheellä. Oppimistulosta häiritsee kuvassa 4.11 ilmenevä lähekkäisten vetovoimakeskusten ongelma. Poikittain kuvapinnan yli juoksevat tiheäviivaiset alueet ovat verkon muodostamia yleistyksiä, joihin kuuluu näytteitä useista peräkkäisistä käyrän huipuista. Ongelman voisi ratkaista pilkkomalla otos x-akselilta useaan erilliseen opetusjoukkoon. Tällöin myös oppiminen nopeutuisi.

Johtopäätöksenä testiajosta on, että käytetyt opetusmenetelmät ja verkkotyypit eivät ole riittäviä monimutkaisissa syöteavaruuksissa. Helpotetulla tehtävällä, jossa luokat ovat avaruudellisesti kauempana toisistaan, oppiminen sujui paremmin (kuvassa 4.11 oikealla).

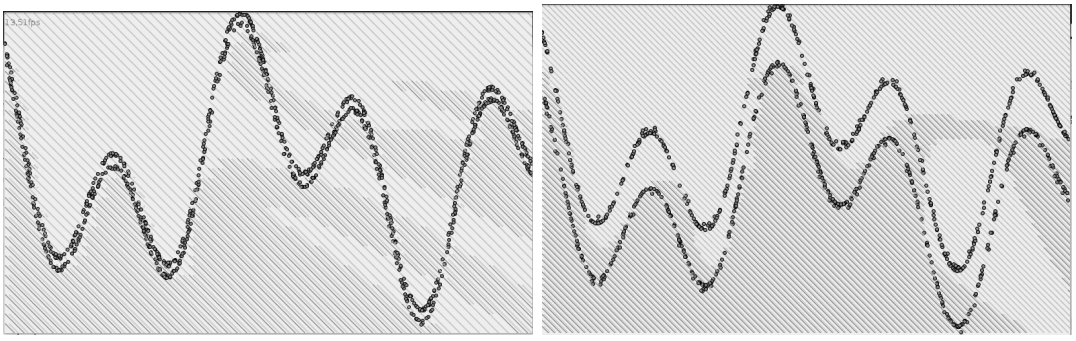
4.3 New-thyroid

New-thyroid-oppimistehtävä on lähes lineaarisesti erottuva. Hyvän erottuvuuden vuoksi aineisto ei vaatinut esikäsittelyä.

Kahden piiloneuronin rakenne pääsi parhaimmillaan varsin hyvään 2.4%:n opetusvirheeseen ja 5.0%:n testivirheeseen. Parhaan oppimistuloksen antoi puolestaan eräs 23x4-rakenne (0%, 1.9%), pienimmän testivirheen eräs 8x4-verkko (1.9%, 0.7%). Näillä hyvillä luokittelijoilla oppimisvarianssi oli pientä tai merkityksetöntä (193.95±



Kuva 4.10 Superimposed-sines-2-oppimistehtävä. Tehtävä on verrattain tiheä näytteiltään ja käyrän vaihtelujen vuoksi oletuksen mukaan vaatii useita piiloneuroneja. Myös ratkaisun supistuminen kohti globaalia ratkaisua voi tuottaa ongelmia. Luokiteltavat arvot tulevat käyrän molemmin puolin tasajakaumasta $U[-5, 5]$ näytteistetystä normaalijakaumasta $N(\mu, \sigma^2)$ parametreilla $\mu = 0$, $\sigma^2 = 4$.

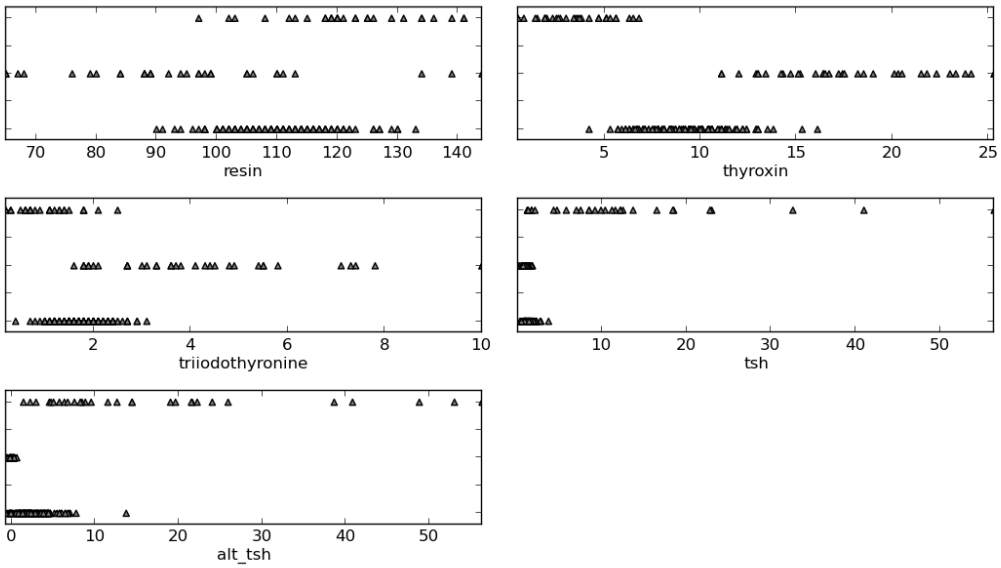


Kuva 4.11 Superimposed-sines-2-oppimistehtävän luokittelu (vasemmalla). Helpotettu tehtävä (oikealla). Luokat merkitty vaalea vinoviivalla ja tummalla, tiheämmällä vinoviivalla.

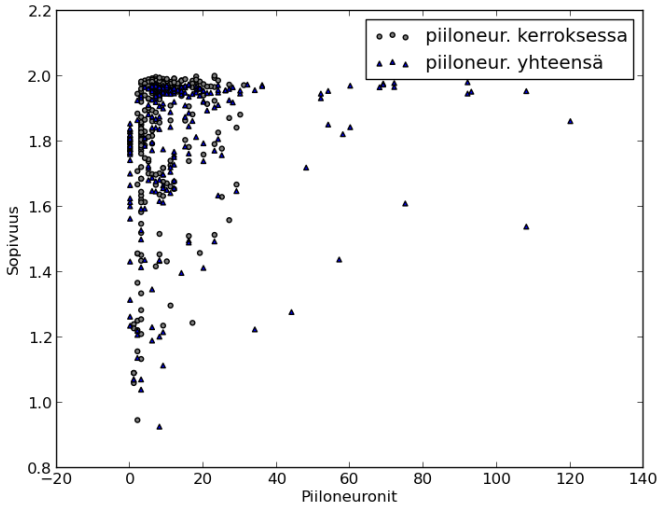
$0.0, 20.6 \pm 0.34; 190.3 \pm 0.41, 20.85 \pm 0.23$). Suositukseksi New-thyroid-tehtävän parhaalla tavalla ratkaisevasta rakenteesta sopii pienimmän testivirheen antanut 8x4-verkko myös luokittelun varianssin ollessa pientä.

Oppimis- ja testituloksen tasaisuutta optimoiva opetusproseduuri aiheutti sen, että opetusjoukkoa ymmärtävä verkko harvoin edusti ajoa, mikäli huomattavaa testivirhettä esiintyi (vrt. kaava 3.3). Toisin sanoen opetusproseduuri ehkäisi ylisovittumista. Samaa havainnollistaa myöhemmin esitettävä loki Haberman-ajosta.

Kuvassa 4.13 esitetään kaikkien testiajojen sopivuuden jakauma piiloneuronien määrän funktiona. Geneettinen algoritmi on keskittänyt hakuaan 6-neuronisen verkon ympärille. Oppimistulos on vaihdellut paljon saman neuronimäärän sisältävien rakenteiden sisällä. Osittain tämä selittyy eri oppimisproseduureilla ja algoritmeilla, mutta osittain myös oppimisalgoritmien vaihtelevalla kyvyllä opettaa syviä neuroverkkoja.



Kuva 4.12 New-thyroid-oppimistehtävä. Otos käsittelee kilpirauhasen suurentuman ennustamista. Näytteen muuttujat on eroteltu kuvassa alakaavioiksi, joissa y-akselilla aina luokkajakauma (normaali, hyper, hypo). Kuvasta ilmenee, että kaikki muuttujat ovat luokkajakauman kannalta merkityksellisiä, joskin kolmen ensimmäisen muuttujan antama erottelu on luotettavampi.



Kuva 4.13 New-thyroid, testiajojen sopivuuden jakauma piiloneuronien määrän funktiona.

4.4 Bupa

Bupa-oppimistehtävä on edellisiä haastavampi monimutkaisemman ja osin sisäisesti ristiriitaisen syötteen ansiosta (kuva 4.14). Erityisesti luokkajakauman variaatio suhteessa drinkit-muuttujaan on suurta. Toisaalta tämä muuttuja näyttää lineaarisen regression pohjalta liittyvän maksasairausennusteeseen. Muuttuja tasoitettiin klusteroimalla se kuuteen (6) tasaväliseen lokeroon.

Tulosta voitiin parantaa myös jonkin verran lisäämällä syötteeseen johdannaiset kuutiolliset alamiiniaminotransferenssia ja aspartaattiaminotransferenssia kuvaavat muuttujat, jotka levittäytyivät alkuperäistä tasaisemmin syöteavaruuteen. Mikään syötemuuttujapari ei mahdollistanut luokkien lineaarista erottelua toisistaan, vaan luokat lomittuivat kaikissa 2-ulotteisissa syöteavaruuksissa toistensa sekaan.

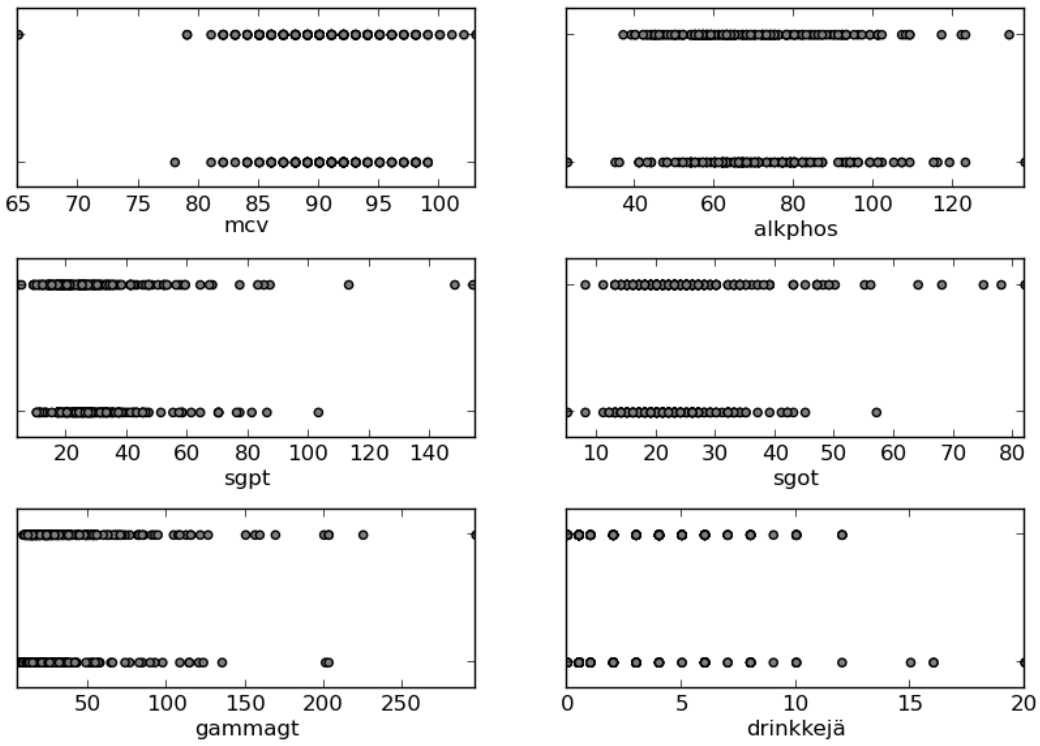
Haun syvyys jäi ohueksi, sillä se tutki vain 171 verkkorakenne–oppimisalgoritmiyhdistelmää ennen määrätyn ajan loppumista.

Mikään lineaarinen luokittelija ei suoriutunut hyvin tehtävästä, parhaan (huonoimman) päästessä 40.9 (51.6)%:n opetusvirheeseen ja 32.1 (51.6)%:n testivirheeseen.

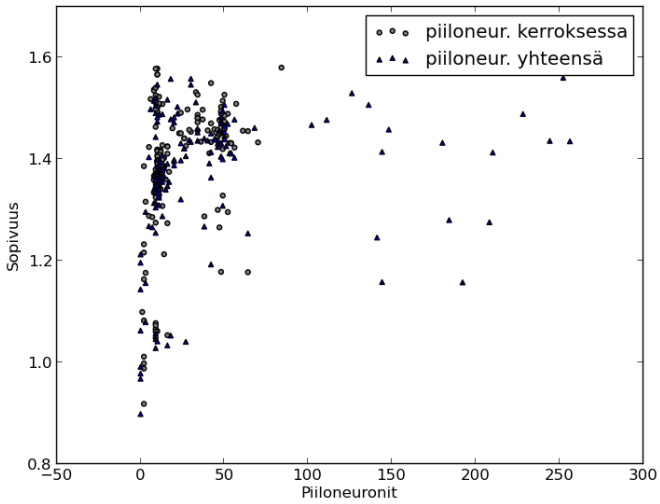
Haku ei osunut yhteenkään 2x1-rakenteeseen, joka ei olisi karsiutunut pois, mutta eräs 2x4-rakenne tuotti 29.7%:n opetusvirheen ja 33.7%:n testivirheen luokitteluvarianssin ollessa erittäin suurta ($218.6 \pm 367.4, 22.6 \pm 13.9$).

Paras luokittelija oli 84x3-verkko 7.1%:n opetus- ja 19.6%:n testivirheellä. Testiluokittelun varianssi oli edelleen merkittävää ja opetusjoukon luokittelun varianssi erittäin suurta ($288.9 \pm 195.8, 27.4 \pm 3.4$). Löytynyt verkko oli neuronimäärältään suurin haun tutkima rakenne. Toiset suuret rakenteet olisivat todennäköisesti vielä parantaneet tulosta. Sopivuutta neuronimäärän funktiona on kuvattu kuvassa 4.15. Haun suurempi syvyys suurikokoisissa verkkorakenteissa olisi suonut viitettä siitä, johtuivatko erittäin suuret varianssit otoksen huonosta jakautumisesta opetus- ja testijoukkoihin vai oppimisalgoritmin epädeterministisyydestä.

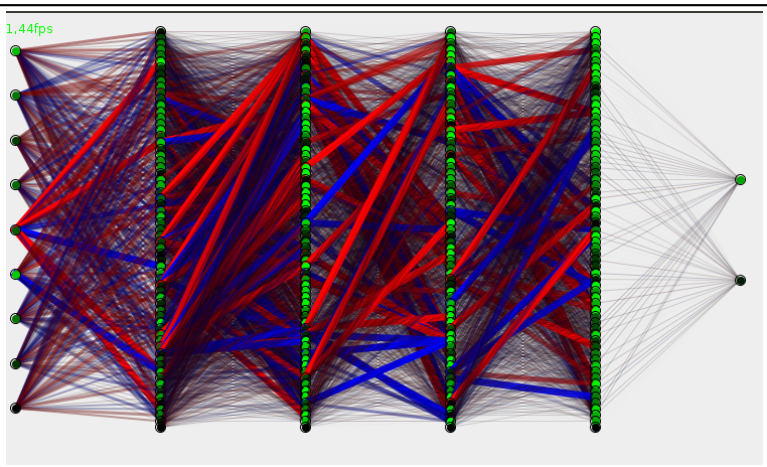
43x4-verkon ajonaikainen visualisaatio kuvassa 4.16 havainnollistaa verkon sisäisen rakenteen eriytymistä opetuksen myötä: osa kaarten painoarvoista on vahvistunut toisten kuihtuessa kohti nollaa (ohut tai näkymätön viiva).



Kuva 4.14 Bupa-oppimistehtävä. Otos käsittelee miesten maksasairauksien ennustamista. Y-akselilla luokkajakauma (maksasairauksia, ei maksasairauksia). X-akselilla alakuvissa aineiston muut muuttujat (lihastilavuus, alkalifosfotaasi, alamiiniaminotransferenssi, aspartaattiaminotransferenssi, gammaglutamiilin transpeptidaasi, drinkit annosta per päivä).



Kuva 4.15 Bupa, testiajojen sopivuuden jakauma piiloneuronien määrän funktiona. Haku on ehtinyt tuottaa vain hajanaisia näytteitä suuremmilla verkkorakenteilla, mutta jakaumassa on nähtävissä heikko nouseva trendi.



Kuva 4.16 Ajonaikainen visualisaatio 43x4-luokittelijasta Bupa-tehtävässä. Syöteneuronit äärimmäisenä vasemmalla ja tulosneuronit (luokittelu) äärimmäisenä oikealla.

4.5 Ecoli

Luokittelutehtävä käsittelee proteiinien paikallistamista bakteerisoluissa (kuva 4.17). Aineisto sisältää seuraavat muuttujat: taltiointiin liittyvä sekvenssin nimi (0), Mc-

Geochin menetelmän signaali sekvenssin tunnistukseen (mcg), von Heijnen menetelmä signaali sekvenssin tunnistukseen (gvh), von Hejnen peptidaasi II konsensussignaali (lip, binäärinen), varauksen olemassaolo ennustettujen lipoproteiinien N-terminuksessa (chg, binäärinen), diskrimanttianalyysin tulos ulomman solukalvon ja periplasmisten proteiinien aminohapposisällöstä (aac), ALOM kalvon läpäisevän alueen ennuste (alm1), ALOM-ennuste oletettavasti lohkaistavien alueiden karsimisen jälkeen (alm2). Binääriset muuttujat lip ja chg käsittävät vain arvoja 0 ja 1, ja näistä chg on lisäksi miltei merkityksetön, sillä kaikki näytteet yhtä lukuunottamatta kuuluvat siinä luokkaan 0. Tämä antaisi muuten aiheen muuttujan karsintaan, mutta korrelaatio harvassa esiintyvän luokkamuuttujan imS kanssa on korkea. Luokkamuuttujat alhaalta ylös ovat (proteiinilähde, frekvenssi): cp (solulima) 143, im (sisempi kalvo ilman signaalisekvenssiä) 77, pp (periplasma) 52, imU (sisempi kalvo, lohkaisematon signaalisekvenssi) 35, om (ulompi kalvo) 20, omL (ulomman kalvon lipoproteiini) 5, imL (sisemmän kalvon lipoproteiini) 2, imS (sisempi kalvo, lohkaistava signaalisekvenssi) 2.

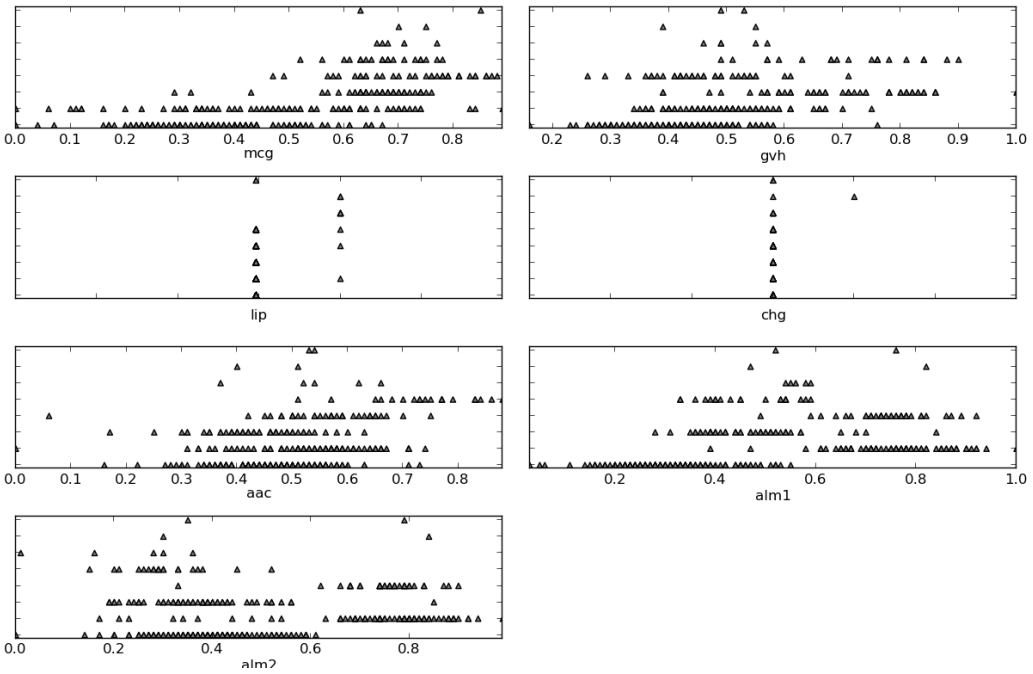
Otokselle tehtiin seuraavat manuaaliset esikäsittelytoimet: karsittiin taltiointiin liittyvä sekvenssin nimi (0). Aineiston automaattisessa esikäsittelyssä normalisoitiin diskreetit muuttujat ja klusteroitiin ja jaettiin usealle syötteelle binääriset muuttujat. Testi- ja opetusjoukkojen muodostaminen poikkesi muista aineistoista siten, että tässä ei voitu käyttää satunnaista luokkaperustaista jakamista opetus- ja testijoukkoihin osan luokkamuuttujista ollessa aliedustettuja (niukka data). Testijoukko arvottiin välittämättä sen saamasta luokkajakaumasta. Tämä vääristää testituloksia parantaen niitä muihin käsiteltyihin aineistoihin verrattuna.

Paras luokittelija oli sigmoid(1) minibatch 16x1-verkko (opetusvirhe 8.6%, testivirhe 8.8%). Variaatio oli testijoukossa kohtalaisen merkittävää ylittäen niukasti 5% rajan ($285.4 \pm 7.9, 29.3 \pm 1.8$). Paras opetusjoukon luokittelija oli sigmoid(9) 16x2-verkko (5.8%, 11.4%). Yksittäisissä ajoissa jotkin verkot pääsivät 3.0%:n opetusvirheeseen ja ajon aikana tätäkin pienempiin opetusvirheisiin, testivirheen jäädessä tällöin kuitenkin tuntuvasti suuremmaksi (ylisovittuminen).

Lineaarinen luokittelija ylsi parhaimmillaan (14.0%, 13.6%) virheeseen suurella $> 10\%$ variaatiolla testijoukossa ($260.7 \pm 1.8, 28.5 \pm 3.2$). Epälineaarinen luokittelija suoriutui siis merkittävästi lineaarista luokittelijaa paremmin. Paras löydetty 2x1-verkko suoriutui toisaalta selvästi parasta lineaarista luokittelijaa kehnommin (opetusvirhe 35.3%, testivirhe 34.1%).

Lohkaisemalla pois imS-signaalisekvenssi ja siihen liittyvä varaus-muuttuja voitiin hyödyntää luokkaperustaista testijoukon jakaumaa. Tällöin päästiin sopivilla luokittelijoilla varsin hyviin virheisiin (4-8%, 8-12%). Standardoimalla käsitellyt luo-

kat (kaava 3.2) jakautuivat yleensä tasaisemmin tulosavaruuden yli, mutta luokittelutulos oli järjestään heikompi kuin minmax-normalisoinnissa.



Kuva 4.17 Ecoli-oppimistehtävä. Näytteen muuttujat on eroteltu alakaavioiksi, joissa y-akselilla luokkajakauma.

4.6 Haberman

Haberman-oppimistehtävä on esitelty kuvassa 4.18. Otos käsittelee syöpäpotilaiden eloonjäämistä 5 vuotta leikkauksen jälkeen. Aineistolle tehtiin seuraavat manuaalisest esikäsittelyvaiheet. Potilaan ikä klusteroitiin kuuteen (6), operointivuosi neljääntoista (14) ja havaittujen etäpesäkkeiden määrä kahteentoista (12) lokeroon. Saadut diskreetit muuttujat pilkottiin automaattisesti kukin useaksi syötemuuttujaksi.

Lineaarinen luokittelija pääsi parhaimmillaan (huonoimmillaan) 21.6 (52.8)%:n opetusvirheeseen ja 26.5 (50.5)%:n testivirheeseen. Luokittelun varianssi oli tällöin varsin suurta: $216.5 \pm 11.6(130.4 \pm 985.4), 22.1 \pm 2.1(14.9 \pm 4.5))$, mikä viittaa ennustamattomuuteen oppimistilanteessa.

Kahden piiloneuronin luokittelija pääsi parhaimmillaan (huonoimmillaan) 19.2 (23.9)%:n opetusvirheeseen ja 32.2 (40.8)%:n testivirheeseen, eli suoriutui vain hieman lineaarista luokittelijaa paremmin. Onnistuneen luokittelun varianssi oli suu-

rempaa kuin lineaarisella luokittelijalla: $223.0 \pm 41.4 (210.0 \pm 8.0), 20.4 \pm 3.6 (17.8 \pm 9.5)$, mikä viittaa vielä suurempiin yksilöllisiin vaihteluihin oppimistuloksissa. Toisaalta parhaimman ja huonoimman tuloksen etäisyys sekä huonoimman tuloksen varianssi olivat pieniä verrattuna lineaarisen luokittelijan vastaaviin. Kahden piiloneuronin rakenne siis näytti tuottavan tasaisempia tuloksia pitkällä tähtäimellä. Tämä selittyy erityisesti sillä, että opetus- ja testijoukon valinnassa esiintyvä satunnainen vaihtelu häiritsi enemmän lineaarisen kuin kahden piiloneuronin luokittelijan parasta mahdollista suoriutumista.

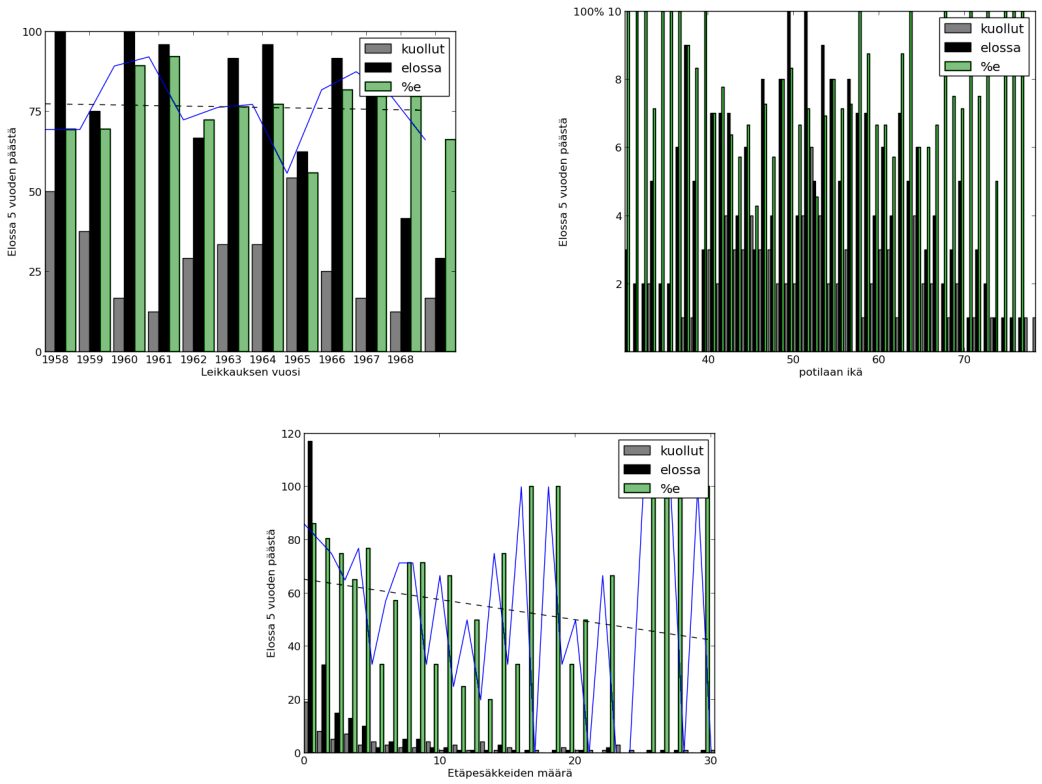
Aineiston haastavuuden takia monimutkaisemmat verkkorakenteet suoriutuivat vain hieman yksinkertaisia verkkorakenteita paremmin. Eräs 22x3-verkko antoi virheen (15.2%, 25.5%). 75%:n onnistumista voidaan pitää kohtalaisen hyvänä tuloksena, kun kohina aineiston sisällä oli varsin suurta ja muuttujat kovin niukat. Parhaan opetusvirheen 14.3% antoi eräs 43x4-verkko, mutta testivirhe oli tällöin vastaavasti kehno 35.7%, eli kyseessä oli selvä ylisovittuminen, jota opetusproseduuri hieman ehkäisi. Mainittakoon että ajojen aikana monet luokittelijat pääsivät tätä pienempiin opetusvirheisiin. Oppimisalgoritmi kuitenkin kehitti ajoja kohti oppimis- ja testivirheen tasapainoa.

Sopivasti klusteroimattomalla datalla opetusvirhe saatiin vieläkin alemmas, mutta testivirhe pysyi vastaavasti hyvin korkeana.

Seuraavassa ajolokin katkelmassa oppimisalgoritmi löytää ensin varsin hyvän luokittelijan opetusjoukolle, mutta valitsee sitten toistuvasti sellaisen, jolla opetus- ja testijoukkojen luokittelut ovat lähempänä toisiaan.

```
tok=16/30 lok=224/276 error=0,78044 lrate=0,10000 it=0
tok=16/30 lok=221/276 error=0,85307 lrate=0,10000 it=1
tok=17/30 lok=235/276 error=0,82319 lrate=0,10000 it=17
tok=17/30 lok=233/276 error=0,81762 lrate=0,10000 it=18
tok=18/30 lok=243/276 error=0,71938 lrate=0,10000 it=31
tok=18/30 lok=242/276 error=0,77753 lrate=0,10000 it=37
tok=18/30 lok=239/276 error=0,76504 lrate=0,10000 it=137
tok=18/30 lok=201/276 error=0,67679 lrate=0,10000 it=174
tok=18/30 lok=220/276 error=0,57305 lrate=0,10000 it=184
tok=18/30 lok=213/276 error=0,62663 lrate=0,10000 it=185
tok=19/30 lok=235/276 error=0,59879 lrate=0,10000 it=334
tok=20/30 lok=243/276 error=0,63250 lrate=0,10000 it=393
tok=20/30 lok=239/276 error=0,59259 lrate=0,10000 it=418
tok=22/30 lok=227/276 error=0,45358 lrate=0,10000 it=478
EBP finished, tok= 22 lok= 227 15042ms, 1479 lrn-epochs
```

Taulukko 4.8: Ajoloki Haberman-oppimistehtävästä.



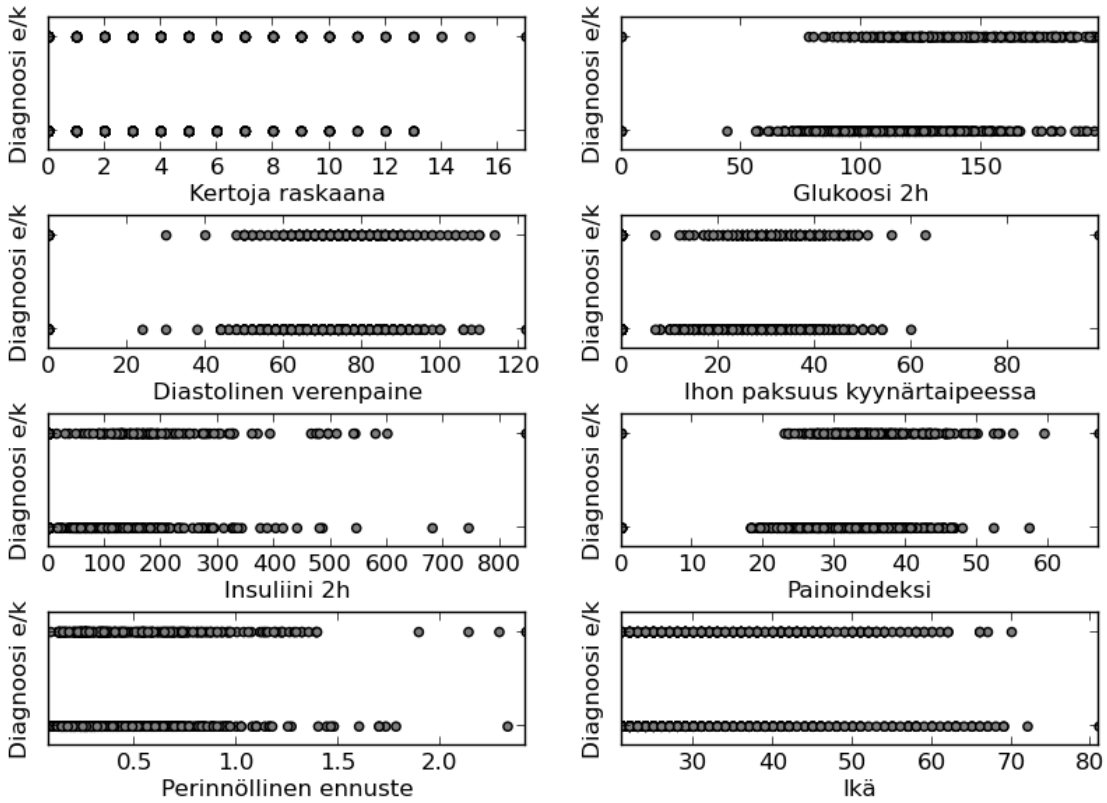
Kuva 4.18 Haberman-oppimistehtävä. Y-akselilla eloonjääneiden määrä. Yhtenäisellä viivalla merkitty eloonjääneiden määrän suhdeluku ja katkoviivalla lineaarinen regressio. Ensimmäisessä kaaviossa, jossa x-akselilla on operaatiovuosi, lineaarinen regressio osoittaa miltei olematonta negatiivista korrelaatiota. Jotkin leikkausvuodet näyttävät potilaan kannalta hyvinkin suotuisilta toisiin verrattuna, mutta taustalla voi olla tekijöitä, jotka tekevät vaihtelusta merkittävän, kuten henkilökunnan vaihtuminen tai ympäristötekijät. Toisessa kaaviossa eloonjääneiden määrä potilaan iän funktiona. Heilahtelu lähekkäisten ikien välillä on suurta ja jatkuvaa, joten iän klusterointi suurempiin luokkiin näyttää mielekkäältä (datan esikäsittely). Kolmannessa kaaviossa eloonjääneet havaittujen etäpesäkkeiden määrän funktiona. Lineaarinen regressio antaa viitteitä negatiivisesta korrelaatiosta. Näytteet harvenevat merkityksettömiksi kohti suurempia etäpesäkkeiden määriä, joten klusterointi on tässäkin paikallaan.

4.7 Pima

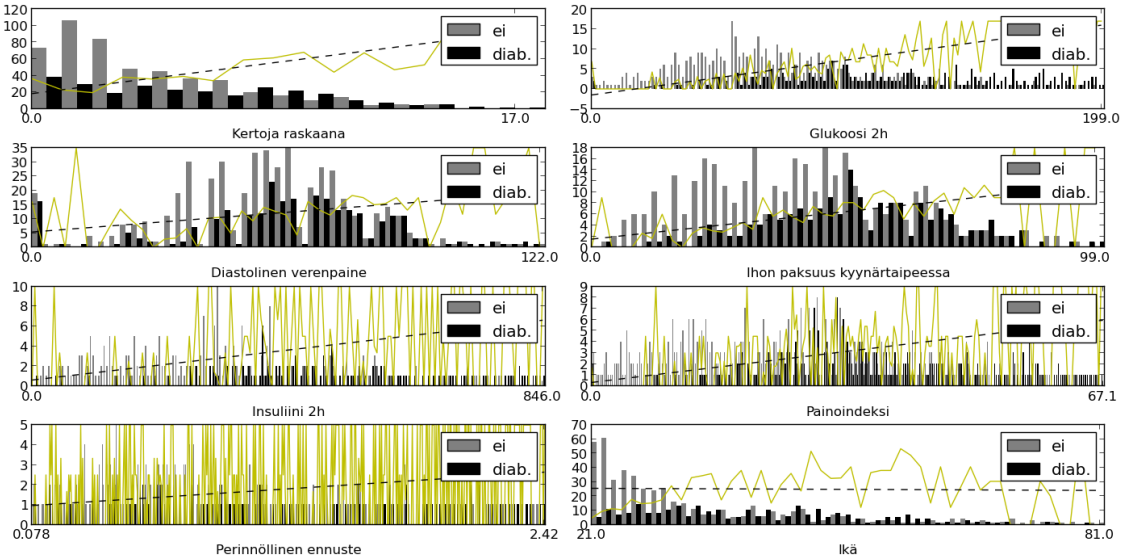
Otos käsittelee aikuisten Pima-intiaaninaisten sokeritaudin ennustamista (kuvat 4.19 ja 4.20). Aineistolle tehtiin seuraavat manuaaliset esikäsittelyvaiheet. Raskauksien

määrä klusteroitiin kuuteen (6) ja potilaan ikä samoin kuuteen (6) lokeroon. Saadut diskreetit muuttujat pilkottiin automaattisesti kukin useaksi syötemuuttujaksi.

Monimutkaisten rakenteiden käyttäminen ei juurikaan parantanut testijoukon luokittelua lineaariseen luokittelijaan verraten. Pienin opetusvirhe oli haussa minibatch-opetetulla 32x1-verkolla (opetusvirhe 12.0%, testivirhe 22.2%) ja pienin testivirhe minibatch-opetetulla 27x1-verkolla (12.1%, 21.5%). Lineaarinen luokittelija pääsi parhaimmillaan (huonoimmillaan) 18.6 (24.4)%:n opetusvirheeseen ja 23.0 (34.0)%:n testivirheeseen. Parhaat luokittelijat pääsivät ajojen aikana huomattavasti pienempiin opetusvirheisiin, mutta käytetty oppimisproseduuri optimoi kohti testi- ja opetusvirheen tasapainoa.



Kuva 4.19 Pima-indians-oppimistehtävä, luokkajakauma syötemuuttujien funktiona. Näytteen piirteet ovat huonoja lineaarisia erottelijoita.



Kuva 4.20 Pima-indians-oppimistehtävä, luokkajakauman frekvenssit. Katkoviivalla merkitty lineaarinen regressio ja yhtenäisellä diabeetikkojen prosentuaalinen osuus.

5 Analyysi

Olen tarkastellut MLP-neuroverkon arkkitehtuurin automaattista optimointia luokittelutehtävissä. Optimoinnin määrittelin hakutehtävänä diskreetissä arkkitehtuuriavaruudessa siten, että haku etsii hyvää verkkorakenne–oppimisalgoritmi-yhdistelmää.

Tutkimusasetelma painotti luokittelun luotettavuutta mittaamalla oppimisvirhettä yhtäältä sekä opetus- että toisaalta testijoukolle, jotka tulevat molemmat samasta otoksesta. Arvioinnissa painotin arkkitehtuurin kykyä luokitella testijoukko oikein, mutta myös opetus- ja testivirheen yhtenevyyttä. Oppimisvirheitä tarkasteltiin mittaamalla keskiarvo useiden oppimissyklien yli. Oppimisvirhettä ja varianssia tarkasteltiin kvalitatiivisesti tutkielman neljännessä luvussa.

Käytetyssä oppimisrutiinissa pyrin minimoimaan testivirheen varianssin. Tätä varten opetus- ja testijoukot näytteistettiin toistuvasti otoksesta siten, että muuten satunnaisesti arvotun testijoukon luokkajakauma noudatti koko otoksen luokkajakaumaa toistojen yli. Menettely tasoittaa testijoukon oppimisvirheen varianssia erityisesti tilanteessa, jossa luokkajakauman varianssi otoksessa on suurta, mutta johtaa välillä hieman verrokkia suurempaan oppimisvirheeseen niukalla datalla (verrokkialgoritmi poimii testijoukon satunnaisesti välittämättä luokkajakaumasta). Arkkitehtuurin optimointitehtävän kannalta oppimisvirheen suuruus ei ole yhtä tärkeä kuin oppimisvirheiden suhteellinen suuruus luokittelujen välillä, mikä puoltaa osaltaan luokkajakaumalla painotetun testijoukon käyttämistä.

Käytetty ahne opetusrutiini tutki oppimisen aikana verkon sopivuutta (kaava 3.3) poimien muistiin sopivimman verkon. Oppimisajon lopuksi sitä sitten käytettiin opetus- ja testivirheen laskennassa. Tämä johti siihen, että saatu opetusvirhe oli helposti suurempi suhteessa testivirheeseen kuin mikä verrokkirutiinilla olisi saatu. Kirjallisuudessa esiintyvä kilpaileva validointimenettely on jakaa otos opetus-, validointi- ja testijoukoiksi, käyttäen validointijoukkoa opetuksen aikana ja testijoukkoa tuloksia arvioitaessa. Käytetty validointirutiini poikkesi tästä menettelytavasta seuraavasti. Erillistä validointijoukkoa käytettäessä tavataan yleensä pysäyttää oppiminen, kun validointijoukon oppimisvirheen kulmakerroin alkaa kasvaa. Käytetyssä rutiinissa puolestaan oppimisen ei tarvinnut pysähtyä validointivirheen alkaessa kasvaa. Näin voitiin jossakin määrin ehkäistä tilannetta, jossa oppiminen pysäytetään ennenaikaisesti validointivirheen paikalliseen minimiin. Sen sijaan rutiinissa määriteltiin kynnysraja (2000 epokkia), jonka pituisen kuivan kauden jälkeen oppimisen katsottiin päättyneen. Käytetty menettely lisäsi oppimissyklien määrää ja oppimisen kestoa verrokkiin nähden.

Geneettinen MLP-arkkitehtuurin oppimisalgoritmi tuotti validoinnin perusteel-

la hyviä tuloksia erityisesti haun ollessa leveä (1/6 suhteessa hakuavaruuden kokoon) ja myös usein yllättävän hyviä tuloksia kapeassa haussa ($< 1/100$). Näin ollen geneettisen hakualgoritmin käyttöä oppimistehtävissä voidaan tulosten perusteella suositella tietyin varauksin.

Kirjallisuudessa on todettu, että laskennan vaatima aika voi muodostua MLP-verkkojen optimointia rajoittavaksi tekijäksi. Sama tulos vahvistui myös useissa tutkielman oppimisajoissa. Kun yksittäisen suuren arkkitehtuurin arviointi kesti parhaimmillaan yli tunnin, ei haulla ollut riittävästi aikaa tutkia hakuavaruutta luotettavasti. Ajallisista rajoituksista mainittakoon lisäksi, että käytetyt esimerkkiaineistot olivat verrattain pieniä suurimman ollessa vain 1000 näytteen kokoinen ja datamäärältä suurimman sisältäessä vain noin 6000 yksittäistä arvoa. Suuremmilla otoksilla käytetty menetelmä olisi ollut jopa käyttökelvoton. Toisaalta suuren laskentatehon aikakaudella laskentaresurssit ovat luultavasti monessa tilanteessa halvempia kuin samaan optimointiin käytetty inhimillinen työaika.

Käytettyjen aineistojen joukko oli siinä mielessä vaillinainen, että se ei sisältänyt yhtään aineistoa, jossa olisi hyvin suuri määrä muuttujia. Esimerkiksi monissa lääketieteellisissä aineistoissa muuttujia on usein jopa tuhansia, jolloin tärkeäksi kysymykseksi nousee muuttujien karsinta ja mahdollisesti tehtävän osioiminen luokkien suhteen.

Suoritusajan ongelmaan löydettiin joitakin lievityskeinoja tutkielman aikana. Haun seurattavuuden parantamiseksi hakutulokset kirjattiin ajon aikana tietokantaan. Geneettiseen algoritmiin sisältyvän suuren satunnaisuuden ansiosta haku voitiin myös tarvittaessa välillä keskeyttää ja aloittaa myöhemmin uudestaan nollapopulaatiosta. Kun tulokset olivat alun perin kapeita, ei duplikaattitietueita havaittavassa määrin syntynyt uudella ajolla.

Haun hitauden lievittämiseksi voidaan ehdottaa muutoksia tutkimusasetelmaan. Tämä pätee erityisesti, jos sovellettavuutta halutaan parantaa. Monet muutoksista voi toteuttaa helposti tutkielman sovelluskoodissa.

1. geneettisessä hakuvaiheessa voidaan tyytyä 2–3 oppimistoistoon, jos parhaat tulokset arvioidaan myöhemmin erikseen

2. suoritukseen voidaan kytkeä hajautettuja laskentayksiköitä käytännössä rajattomasti

3. huonoja rakenteita voidaan leikata vielä käytettyä 25%:n opetusvirhettä matalammalta

4. oppiminen voidaan keskeyttää testivirheen kasvavan kulmakertoimen perusteella

Lisäksi voidaan kehittää hakurutiiniin oppimisen kulmakertoimen tarkastelua epokkien yli. Mikäli tutkittu rakenne–oppimisalgoritmi-pari on merkittävästi hitaampi oppija kuin verrokit, sen antama lopullinen tulos harvoin vastaa nopeiden verrokkien tulosta. Tässä tilanteessa parin hylkääminen voi olla paikallaan.

Tämän lisäksi laskennan vieminen GPU-yksiköille voisi nopeuttaa oppimista hyvin merkittävästi.[1] Tutkielman lähdekoodi on suunniteltu melko datasentrisesti suosimaan yksinkertaisia tietorakenteita kuten taulukkoja ja matriiseja, mikä puolestaan helpottaa GPU-migraatiota. Erityisesti minibatch-ajojen vieminen GPU:lle voi olla menestyksellistä, kun suurempi määrä opetusdataa voidaan siirtää näytönohjaimen prosessoitavaksi kerralla.

Todettakoon myös, että käytetty geneettinen algoritmi ei välttämättä ole tyydyttävin mahdollinen hakualgoritmi arkkitehtuuripopulaatiossa, vaan ihmismäisempi konstruktiivinen algoritmi voisi ratkaista hakuongelman tehokkaammin. Tällaisessa haussa tulisi hyödyntää hyvien rakenteiden haarukointia. Hyvä konstruktiivinen algoritmi etsisi esimerkiksi sopivaa piiloneuronien määrää gridihaulla kolmi-, neli- ja viisikerroksisissa verkoissa erikseen. Gridihaulle voisi määritellä 2–3 haarukkaväliä, esimerkiksi 200-näytteisessä otoksessa haarukat 30, 5 ja 2 siten, että tiheämpi haarukka ajetaan vain lähinnä harvemman haarukan parhaiden piikkivälien sisään ja lisäksi vielä pieniä verkkorakenteita käsittävässä ensimmäisessä suuressa piikkivälissä. Suurimmassa haarukassa ajettaisiin kaikki oppimisalgoritmit ja tiheämmissä haarukoissa vain ympärillä parhaiten menestyneet. Vain haun löytämä paras 5-10% varmistettaisiin lopuksi useilla oppimistoistoilla.

Myös tutkielmassa käytettyjä otoksen esikäsittelykeinoja voisi kehittää monin tavoin. Kirjallisuudessa esiintyy runsaasti erilaisia numeerisia ja tilastollisia menetelmiä, joista useat soveltuvat yleisesti koneoppimiaineistojen esikäsittelyyn. Esimerkiksi Lorena ja muut [2012] tutkivat esikäsittelyä tukivektorikoneella (SVM) luokiteltaville syöpädiagnoosiaineistoille, eritellen omiksi vaiheikseen ensiksi alustavan esikäsittelyn, jossa normalisoidaan data ja hylätään puuttuvaa dataa sisältävät näytteet, toiseksi muuttujien valinnan ja kolmanneksi luokittelumallien johtamisen. Lorena ja muut jakavat useita luokkia sisältävän luokitteluongelman aina useaksi binääriseksi ongelmaksi, joissa positiivista tulosta edustaa yksi luokka ja negatiivista tulosta kaikki muut luokat. Esikäsittelyssä etsitään tilastollisesti kutakin luokkaa

[1] Java-projekti Rootbeer (`http://fixme.com`) tarjoaa GPU-laskentaan toimivan kehityspolun.

parhaiten edustavat muuttujat syöteavaruudesta. Lisäksi näytteitä karsitaan mikrotehtäväkohtaisesti sen perusteella, sisältävätkö ne puuttuvia arvoja enemmän kuin ennalta määrätty kynnysarvo (10%), ja yksittäisiä puuttuvia arvoja täydennetään satunnaisesti muuttujan jakaumasta. Luokittelumallit Lorenalla ja muilla [2012] tarkoittavat malleja, jotka muodostavat lopullisen luokittelun binääristen luokittelijoiden avulla. Juhola ja Siermala [2012] tutkivat puolestaan muuttujien karsimista runsasmuuttujaisesta aineistosta uudella hajontametodilla, jonka etuna voidaan pitää hyvää määrittyvyyttä erilaisten muuttujatyyppien yli.

Viime vuosina monineuroverkkojärjestelmien hyödyntäminen on ollut suhteellisen yleistä neurolaskennassa. Eräs esikäsittelyn muoto voisi olla otoksen ja sen muuttujien osioiminen alaverkoille, joiden yhteisen tuloksen perusteella muodostettaisiin ennuste (vrt. [Shields & Casey, 2008]). Erityisesti regressiotehtävissä ja niitä muistuttavissa tehtävissä kuten generoidussa Superimposed sines -aineistossa yksinkertaisella menettelyllä saavutettaisiin tuntuvia etuja sekä luokitteluvoiman että suoritusajan muodossa. Yleisemmin tilanteissa, joissa jokin luokka samaistuu hyvin vahvasti tiettyihin aineiston muuttujiin, tulisi automaattinen ositus kyseeseen. Yksittäinen neuroverkko voidaan yleisemmin nähdä osana laajempaa signaalinkäsittelyjärjestelmää. Tämänkaltainen lähestymistapa on auttanut esimerkiksi ihmisaivojen monimutkaisten ja nopeiden kielenkäsittelyprosessien mallintamisessa [Harley, 2001].

6 Neur-kirjaston viitteet

	Neuroverkon toteutus ja rakenne
J-NEU	https://github.com/pvto/neur/blob/master/neur/Neuron.java
J-MLP	https://github.com/pvto/neur/blob/master/neur/MLP.java
J-ACF	https://github.com/pvto/neur/blob/master/neur/struct/ActivationFunction.java
	Aktivaatiofunktiot
J-LIN	https://github.com/pvto/neur/blob/master/neur/struct/LinearFunc.java
J-SIG	https://github.com/pvto/neur/blob/master/neur/struct/SigmoidalFunc.java
J-SSG	https://github.com/pvto/neur/blob/master/neur/struct/SoftsignFunc.java
J-TNH	https://github.com/pvto/neur/blob/master/neur/struct/TanhFunc.java
	Luokittelu
J-SFM	https://github.com/pvto/neur/blob/master/neur/learning/clf/Fast1OfNClassifier.java
	Datan esikäsittely
J-SMP	https://github.com/pvto/neur/blob/master/neur/auto/in/Sampler.java
J-ARF	https://github.com/pvto/neur/blob/master/neur/util/Arrf.java
	Otos, opetus- ja testijoukot
J-DS	https://github.com/pvto/neur/blob/master/neur/data/Dataset.java
J-TS	https://github.com/pvto/neur/blob/master/neur/data/TrainingSet.java
J-TRM	https://github.com/pvto/neur/blob/master/neur/data/TrainMode.java
J-SYS	https://github.com/pvto/neur/blob/master/neur/data/slice/RandomSlicing.java
J-TRR	https://github.com/pvto/neur/blob/master/neur/data/Trainres.java
	Esiopetus
J-TBS	https://github.com/pvto/neur/blob/master/neur/learning/learner/TabooBoxSearch.java
	Oppimisalgoritmit
J-EBP	https://github.com/pvto/neur/blob/master/neur/learning/learner/BackPropagation.java
J-ELA	https://github.com/pvto/neur/blob/master/neur/learning/learner/ElasticBackProp.java
J-MOM	https://github.com/pvto/neur/blob/master/neur/learning/learner/MomentumEBP.java
J-LRN	https://github.com/pvto/neur/tree/master/neur/learning/learner
	Haku (rakenteen optimointi)
J-MSS	https://github.com/pvto/neur/blob/master/neur/auto/SearchSpaceForClassifierMLPs.java
J-BFS	https://github.com/pvto/neur/blob/master/neur/auto/routine/BruteForceMLPSearch.java
J-GNS	https://github.com/pvto/neur/blob/master/neur/auto/routine/GeneticMLPSearch.java
	Visualisaatiot
J-CVI	https://github.com/pvto/neur/blob/master/neur/util/visuals/ClfVisualisation.java
J-MVI	https://github.com/pvto/neur/blob/master/neur/util/visuals/MLPVisualisation.java

Taulukko 6.1: Java-lähdekoodi

Viiteluettelo

[Ackley & Hinton, 1985] David H. Ackley, Geoffrey E. Hinton & Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive Science* 9 (1985) 147–169.

[Anthony, 2001] Martin Anthony. *Discrete Mathematics of Neural Networks, Selected Topics*. SIAM, Philadelphia, 2001.

[Glorot & Bengio, 2010] X. Glorot, Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *AISTATS* (2010) 249–256.

[Bergstra & al., 2009] Bergstra, J., Desjardins, G., Lamblin, P., & Bengio, Y. Quadratic polynomials learn better image features (Technical Report 1337). *Proceedings of NAACL HLT: Short Papers* (2009) 245–248.

[Babu & Suresh, 2012] G. Sateesh Babu & S. Suresh. Meta-cognitive neural network for classification problems in a sequential learning framework. *Neurocomputing* 81 (2012) 86–96.

[Ballabio et al., 2011] Davide Ballabio, Mahdi Vasigh, Viviana Consonni & Mohsen Kompany-Zareh. Genetic algorithms for architecture optimisation of counter-propagation artificial neural networks. *Chemometrics and Intelligent Laboratory Systems* 105 (2011) 56–64.

[Bechtel & Abrahamsen, 2002] William Bechtel & Adele Abrahamsen. *Connectionism and the Mind. Second Edition*. Basil Blackwell, Cambridge, Massachusetts, 2002.

[Chandra et al., 2012] Rohitash Chandra, Marcus Frean & Mengjie Zhang. On the issue of separability for problem decomposition in cooperative neuro-evolution. *Neurocomputing* 87 (2012) 33–40.

[Chen & Soo, 1995] Hown-Wen Chen, Von-Wun Soo. The plasticity of feedforward neural networks in assimilating a training instance based on non-batch learning. *Neurocomputing* 7 (1995) 299–309.

[Clancey, 1997] William J. Clancey. *Situated Cognition*. Cambridge University Press, Cambridge, 1997.

[Duda et al., 2001] Richard O. Duda, Peter E. Hart & David G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., New York, 2001.

[Erhan et al., 2009] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio & Pascal Vincent. The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training. *AISTATS* (2009) 153–160.

[Hinton et al., 2006] Hinton, G. E., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation* 18 (2006) 1527–1554.

[Hopfield, 1982] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences* 81 (1982) 2554–2558.

[Hyvärinen & Bingham, 2003] Aapo Hyvärinen & Ella Bingham. Connection between multilayer perceptrons and regression using independent component analysis. *Neurocomputing* 50 (2003) 211–222.

[Juhola, 2012] Martti Juhola. *Neurolaskenta. Luentomateriaali.* Unpublished, Tampere University, 2012.

[Juhola & Siermala, 2012] Martti Juhola & Markku Siermala. A scatter method for data and variable importance evaluation. *Integrated Computer-Aided Engineering* 19 (2012) 137–149.

[Kohonen, 1982] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics* 43 (1982) 59–69.

[Kolmogorov, 1957] A. N. Kolmogorov. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. *Biological Dokl. Akad. Nauk. SSSR* 114 (1957) 953–956.

[Lee, 1991] Tsu-Chang Lee. *Structure Level Adaptation for Artificial Neural Networks*. Kluwer Academic Publishers, Norwell, 1991.

[Lorena & al., 2012] Ana C. Lorena, Ivan G. Costa, Newton Spola & Marcilio C.P. de Souto. *Analysis of complexity indices for classification problems: Cancer gene expression data. Neurocomputing* 75 (2012) 33–42.

[Maniezzo, 1994] V. Maniezzo. Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transaction on Neural Networks* 5 (1994) 39–53.

[Minsky & Papert, 1969] Marvin Minsky & Seymour Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1969.

[Mitchel, 1997] T. Mitchel (ed.). *Machine Learning.* MacGraw Hill, New York, 1997.

[Oh & Pedrycz, 2005] Sung-Kwun Oh & Witold Pedrycz. A new approach to self-organizing fuzzy polynomial neural networks guided by genetic optimization. *Physics Letters A* 345 (2005) 88–100.

[Pernía-Espinoza et al., 2005] Alpha V. Pernía-Espinoza, Joaquín B. Ordieres-Mere, Francisco J. Martínez-de-Pisón and Ana González-Marcos. TAO-robust backpropagation learning algorithm. *Neural Networks* 18 (2005) 191–204.

[Puma-Villanueva et al., 2012] Wilfredo J. Puma-Villanueva, Eurípedes P. dos Santos & Fernando J. Von Zuben. A constructive algorithm to synthesize arbitrarily connected feedforward neural networks. *Neurocomputing* 75 (2012) 14–32.

[Rissanen, 1996] J. Rissanen. Stochastic complexity – an introduction. A. Gammerman (ed.): *Computational Learning and Propabilistic Reasoning.* John Wiley & Sons, West Sussex, 1996, 33–42.

[Rumelhart et al., 1986] D. E. Rumelhart, G. E. Hinton & R. J. Williams. Learning internal representations by error propagation. *Parallel distributed processing* 1 (1986) 318–362.

[Rumelhart & McClelland, 1986] D. E. Rumelhart, J. L. McClelland & PDP research group. *Parallel distributed processing: Explorations in the microstructure of cognition. Volumes I-II.* MIT Press, Cambridge, MA, 1986.

[Russell & Norvig, 2009] Stuart J. Russell & Peter Norvig. *Artificial Intelligence: a Modern Approach.* Prentice Hall, London, 2009 (3rd Ed.).

[Rynkiewicz, 2012] J. Rynkiewicz. General bound on overfitting for MLP regression models. *Neurocomputing* 90 (2012) 106–110.

[Sarkar, 1995] Dilip Sarkar. Methods to speed up error back-propagation learning algorithm. *ACM Computing Surveys* 27 (1995) 519–542.

[Schmidhuber et al., 2007] Jurgen Schmidhuber, Daan Wierstra, Matteo Gagliolo & Faustino Gomez. Training Recurrent Networks by Evolino. *Neural Computation* 19 (2007) 757–779.

[Sexton & al., 1995] R.S. Sexton, B. Alidaee, R.E. Dorsey & J.D. Johnson. Global optimization for artificial neural networks: a tabu search application. *European Journal of Operational Research* 106 (1998) 570–84.

[Shao, 2012] Yuanfu Shao. Existence of exponential periodic attractor of BAM neural networks with time-varying delays and impulses. *Neurocomputing* 93 (2012) 1–9.

[Shields & Casey, 2008] Mike W. Shields & Matthew C. Casey. A theoretical framework for multiple neural network systems. *Neurocomputing* 71 (2008) 1462–1476.

[Sivanandam & Deepa, 2008] S. N. Sivanandam & S. N. Deepa. *Introduction to Genetic Algorithms.* Springer, Berlin, 2008.

[Harley, 2001] Trevor Harley. *The Psychology of Language: from Data to Theory.* Psychology Press LTD, Hove, 2001.

[Tosic, 2004] Predrag T. Tosic. A perspective on the future of massively parallel computing. *Proc. of the 1st Conference on Computing Frontiers 2004.* 488–502.

[Valiant, 1984] L.G. Valiant. A theory of the learnable. *Communications of the ACM* 27 (1984) 1134–1142.

[Vapnik, 1996] V. Vapnik. Structure of statistical learning theory. A. Gammerman (ed.): *Computational Learning and Probabilistic Reasoning.* John Wiley & Sons, West Sussex, 1996, 3–32.

[Yang & Chen, 2012] Shih-Hung Yang & Yon-Ping Chen. An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications. *Neurocomputing* 86 (2012) 140–149.

[Wasserman, 1989] Philid D. Wasserman. *Neural Computing.* Springer, Berlin, 1989.